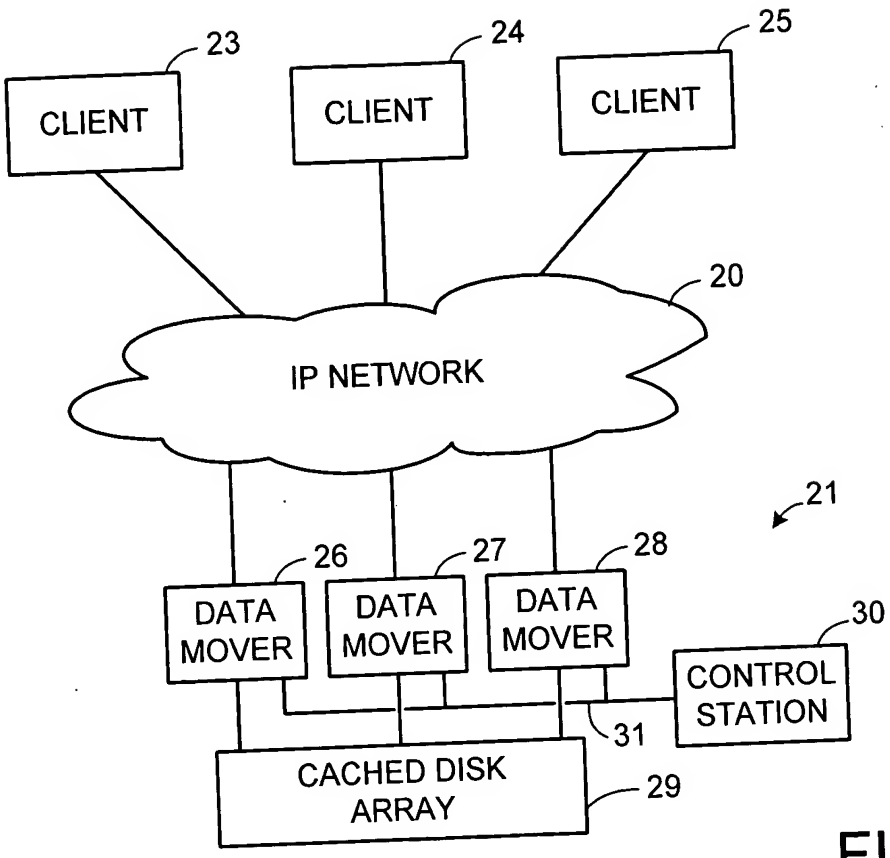

23
24
25
CLIENT
CLIENT
CLIENT
20
IP NETWORK
21
26
27
28
DATA MOVER
DATA MOVER
DATA MOVER
30
CONTROL STATION
31
CACHED DISK ARRAY
29

FIG. 1

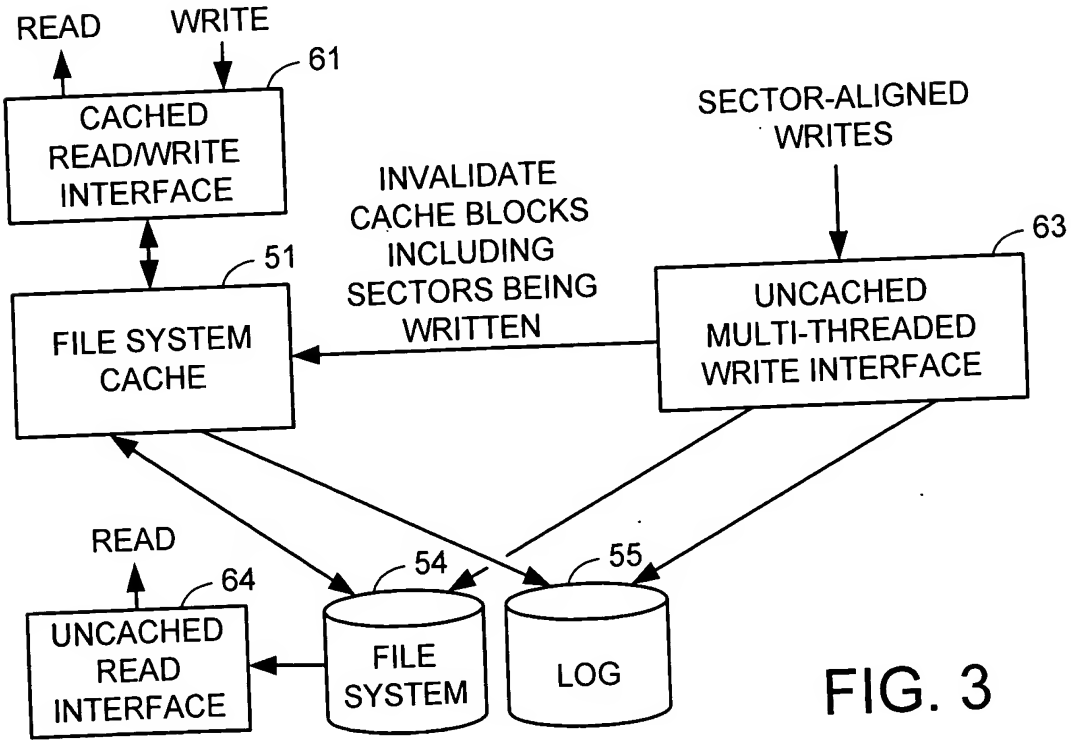

READ
WRITE
61
CACHED READ/WRITE INTERFACE
SECTOR-ALIGNED WRITES
INVALIDATE CACHE BLOCKS INCLUDING SECTORS BEING WRITTEN
63
51
FILE SYSTEM CACHE
UNCACHED MULTI-THREADED WRITE INTERFACE
READ
64
54
55
UNCACHED READ INTERFACE
FILE SYSTEM
LOG

FIG. 3

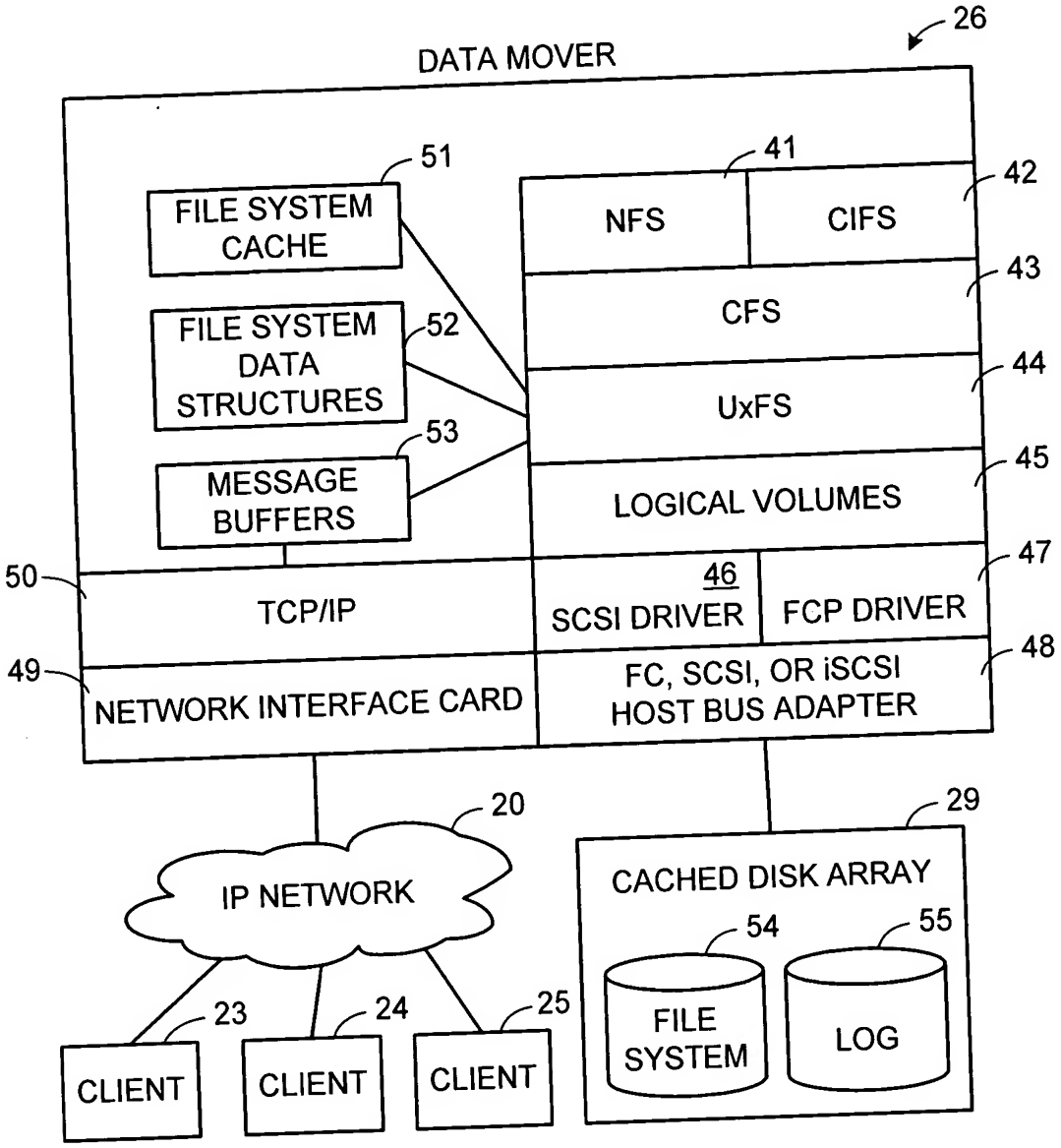

26
DATA MOVER
51
FILE SYSTEM CACHE
52
FILE SYSTEM DATA STRUCTURES
53
MESSAGE BUFFERS
41
NFS
CIFS
42
43
CFS
44
UxFS
45
LOGICAL VOLUMES
47
46
SCSI DRIVER
FCP DRIVER
50
TCP/IP
48
49
NETWORK INTERFACE CARD
FC, SCSI, OR iSCSI HOST BUS ADAPTER
20
IP NETWORK
29
CACHED DISK ARRAY
54
55
FILE SYSTEM
LOG
23
24
25
CLIENT
CLIENT
CLIENT

FIG. 2

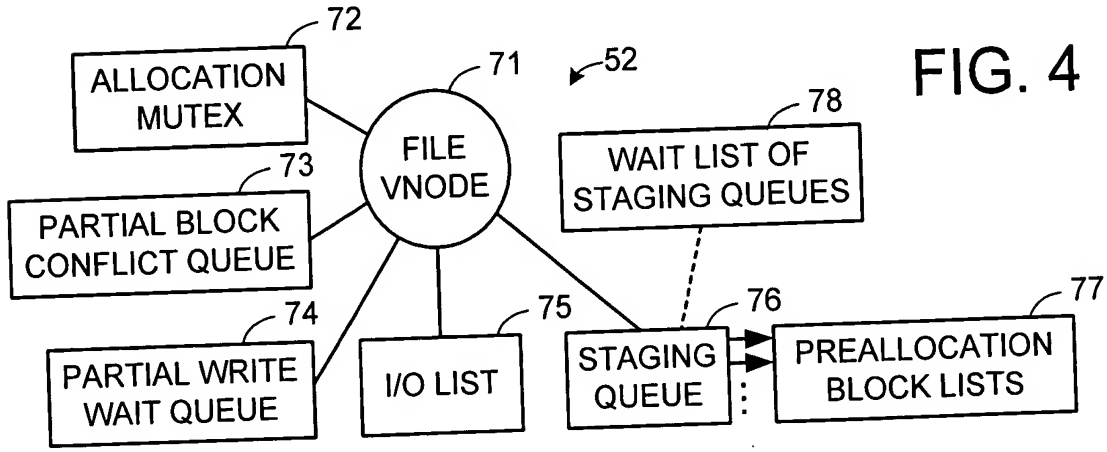

72
ALLOCATION MUTEX
71
52
FILE VNODE
78
WAIT LIST OF STAGING QUEUES
73
PARTIAL BLOCK CONFLICT QUEUE
74
PARTIAL WRITE WAIT QUEUE
75
I/O LIST
76
STAGING QUEUE
77
PREALLOCATION BLOCK LISTS

FIG. 4

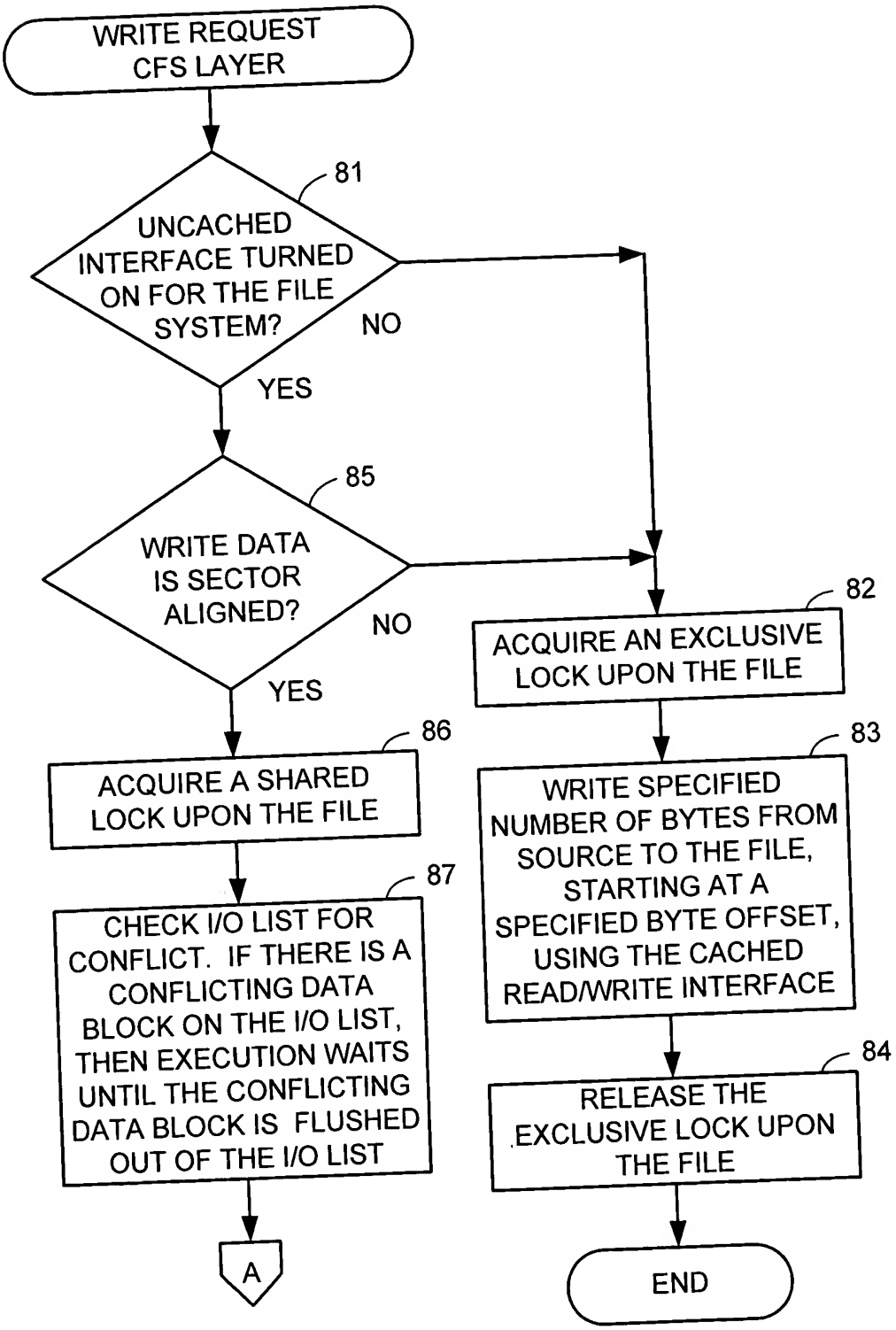

WRITE REQUEST CFS LAYER
81
UNCACHED INTERFACE TURNED ON FOR THE FILE SYSTEM?
NO
YES
85
WRITE DATA IS SECTOR ALIGNED?
NO
YES
82
ACQUIRE AN EXCLUSIVE LOCK UPON THE FILE
86
ACQUIRE A SHARED LOCK UPON THE FILE
87
CHECK I/O LIST FOR CONFLICT. IF THERE IS A CONFLICTING DATA BLOCK ON THE I/O LIST, THEN EXECUTION WAITS UNTIL THE CONFLICTING DATA BLOCK IS FLUSHED OUT OF THE I/O LIST
A
83
WRITE SPECIFIED NUMBER OF BYTES FROM SOURCE TO THE FILE, STARTING AT A SPECIFIED BYTE OFFSET, USING THE CACHED READ/WRITE INTERFACE
84
RELEASE THE EXCLUSIVE LOCK UPON THE FILE
END

FIG. 5

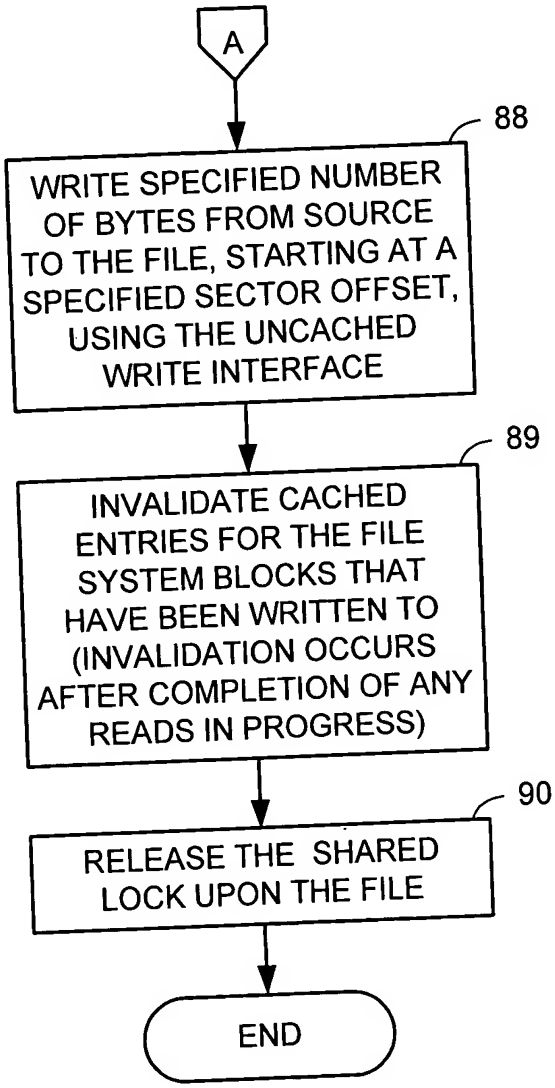

A
88
WRITE SPECIFIED NUMBER OF BYTES FROM SOURCE TO THE FILE, STARTING AT A SPECIFIED SECTOR OFFSET, USING THE UNCACHED WRITE INTERFACE
89
INVALIDATE CACHED ENTRIES FOR THE FILE SYSTEM BLOCKS THAT HAVE BEEN WRITTEN TO (INVALIDATION OCCURS AFTER COMPLETION OF ANY READS IN PROGRESS)
90
RELEASE THE SHARED LOCK UPON THE FILE
END

FIG. 6

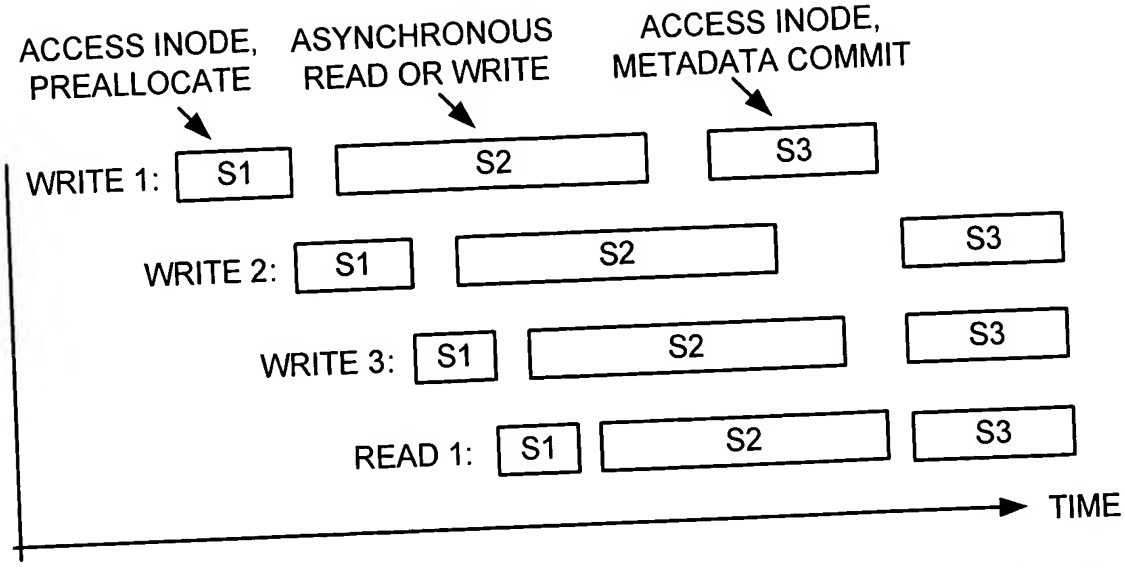

ACCESS INODE, PREALLOCATE
ASYNCHRONOUS READ OR WRITE
ACCESS INODE, METADATA COMMIT
WRITE 1:
S1
S2
S3
WRITE 2:
S1
S2
S3
WRITE 3:
S1
S2
S3
READ 1:
S1
S2
S3
TIME

FIG. 7

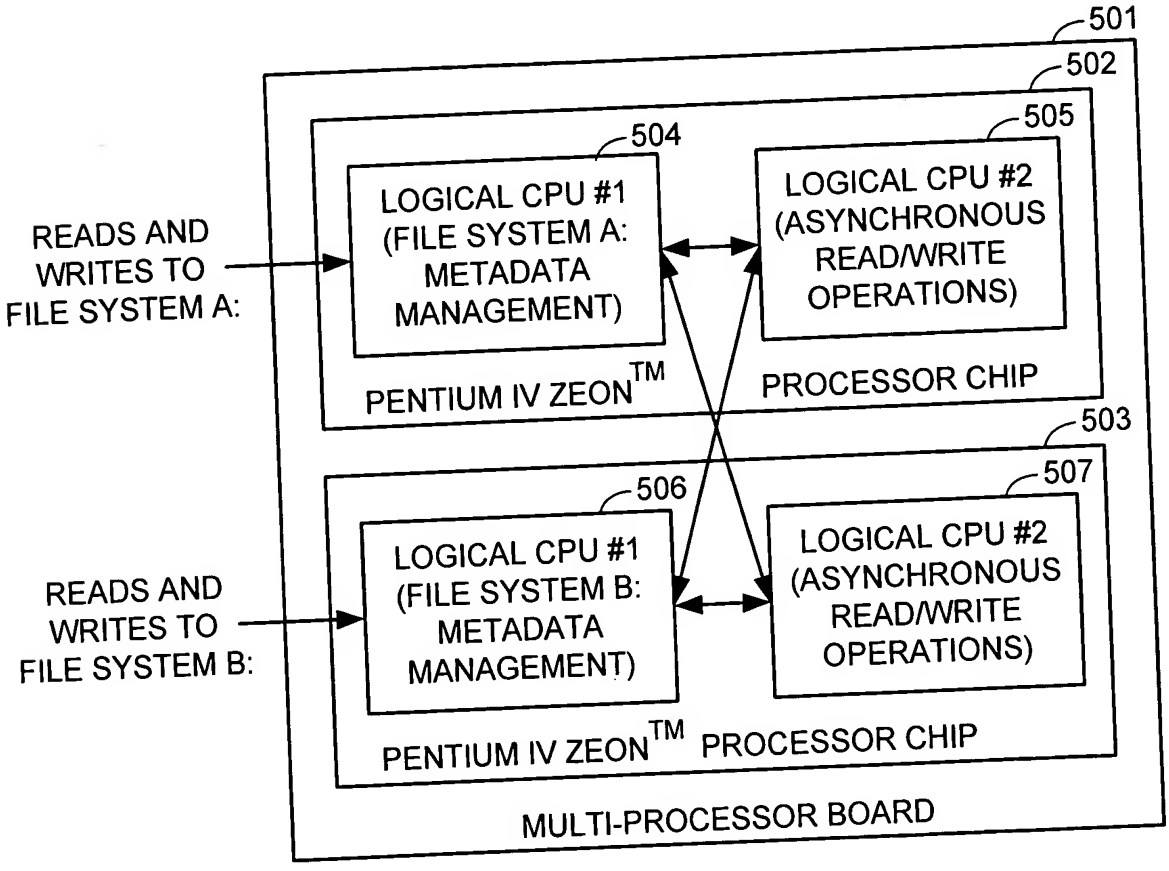

501
502
504
505
READS AND WRITES TO FILE SYSTEM A:
LOGICAL CPU #1 (FILE SYSTEM A: METADATA MANAGEMENT)
LOGICAL CPU #2 (ASYNCHRONOUS READ/WRITE OPERATIONS)
PENTIUM IV ZEON™ PROCESSOR CHIP
503
506
507
READS AND WRITES TO FILE SYSTEM B:
LOGICAL CPU #1 (FILE SYSTEM B: METADATA MANAGEMENT)
LOGICAL CPU #2 (ASYNCHRONOUS READ/WRITE OPERATIONS)
PENTIUM IV ZEON™ PROCESSOR CHIP
MULTI-PROCESSOR BOARD

FIG. 8

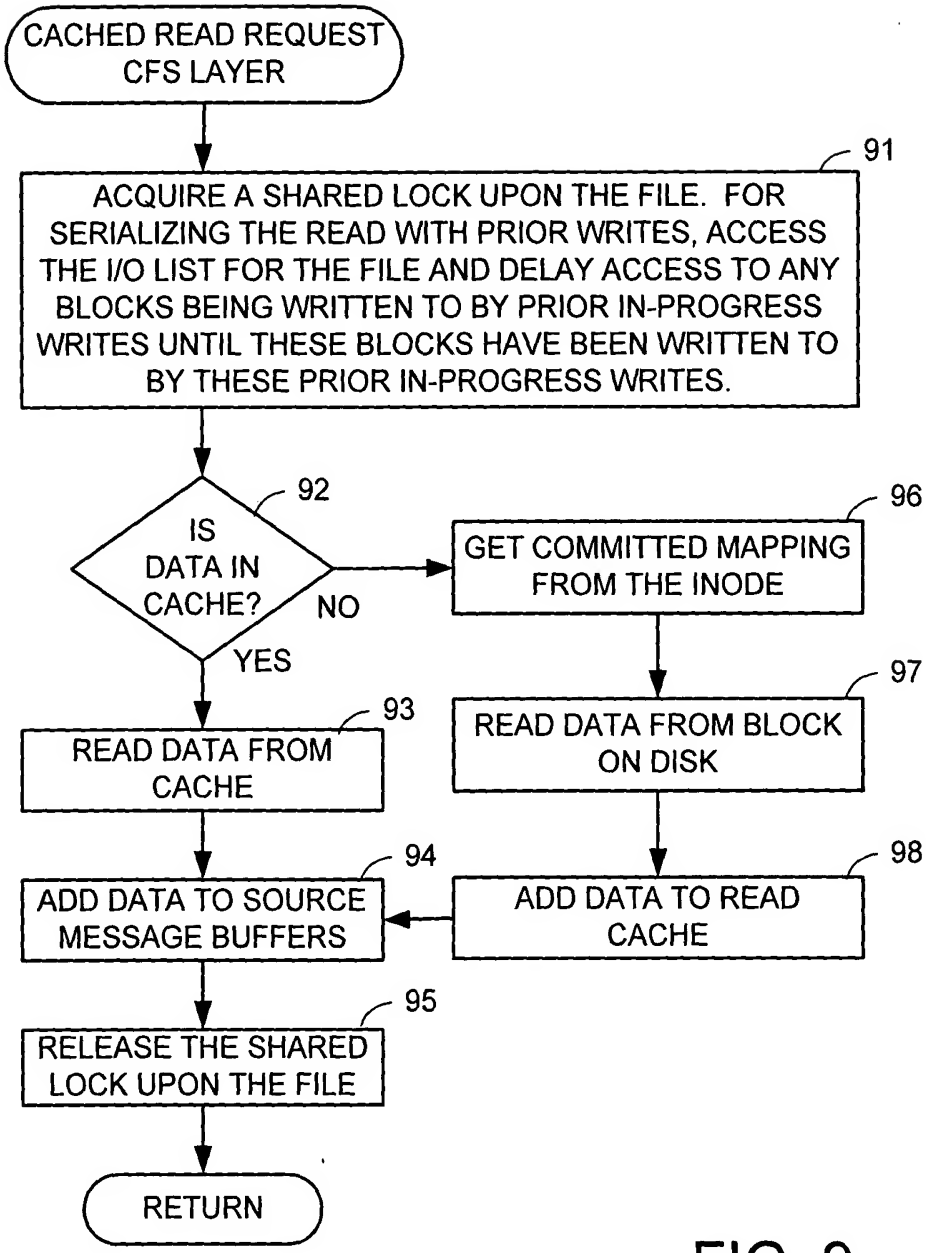

CACHED READ REQUEST CFS LAYER
91
ACQUIRE A SHARED LOCK UPON THE FILE. FOR SERIALIZING THE READ WITH PRIOR WRITES, ACCESS THE I/O LIST FOR THE FILE AND DELAY ACCESS TO ANY BLOCKS BEING WRITTEN TO BY PRIOR IN-PROGRESS WRITES UNTIL THESE BLOCKS HAVE BEEN WRITTEN TO BY THESE PRIOR IN-PROGRESS WRITES.
92
IS DATA IN CACHE?
NO
YES
96
GET COMMITTED MAPPING FROM THE INODE
97
READ DATA FROM BLOCK ON DISK
93
READ DATA FROM CACHE
94
ADD DATA TO SOURCE MESSAGE BUFFERS
98
ADD DATA TO READ CACHE
95
RELEASE THE SHARED LOCK UPON THE FILE
RETURN

FIG. 9

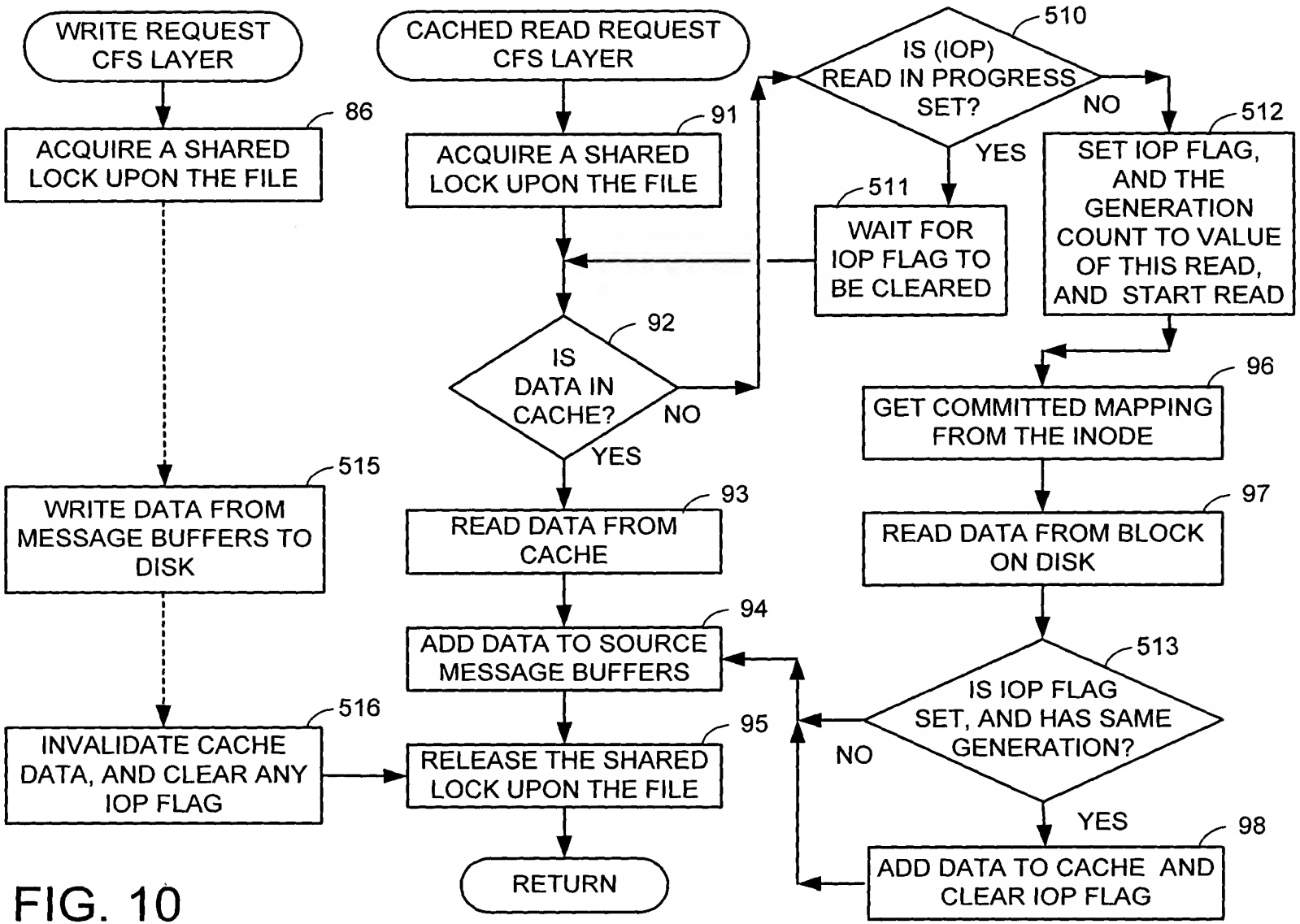

WRITE REQUEST CFS LAYER
86
ACQUIRE A SHARED LOCK UPON THE FILE
515
WRITE DATA FROM MESSAGE BUFFERS TO DISK
516
INVALIDATE CACHE DATA, AND CLEAR ANY IOP FLAG
CACHED READ REQUEST CFS LAYER
91
ACQUIRE A SHARED LOCK UPON THE FILE
92
IS DATA IN CACHE?
NO
YES
93
READ DATA FROM CACHE
94
ADD DATA TO SOURCE MESSAGE BUFFERS
95
RELEASE THE SHARED LOCK UPON THE FILE
RETURN
510
IS (IOP) READ IN PROGRESS SET?
YES
NO
511
WAIT FOR IOP FLAG TO BE CLEARED
512
SET IOP FLAG, AND THE GENERATION COUNT TO VALUE OF THIS READ, AND START READ
96
GET COMMITTED MAPPING FROM THE INODE
97
READ DATA FROM BLOCK ON DISK
513
IS IOP FLAG SET, AND HAS SAME GENERATION?
NO
YES
98
ADD DATA TO CACHE AND CLEAR IOP FLAG

FIG. 10

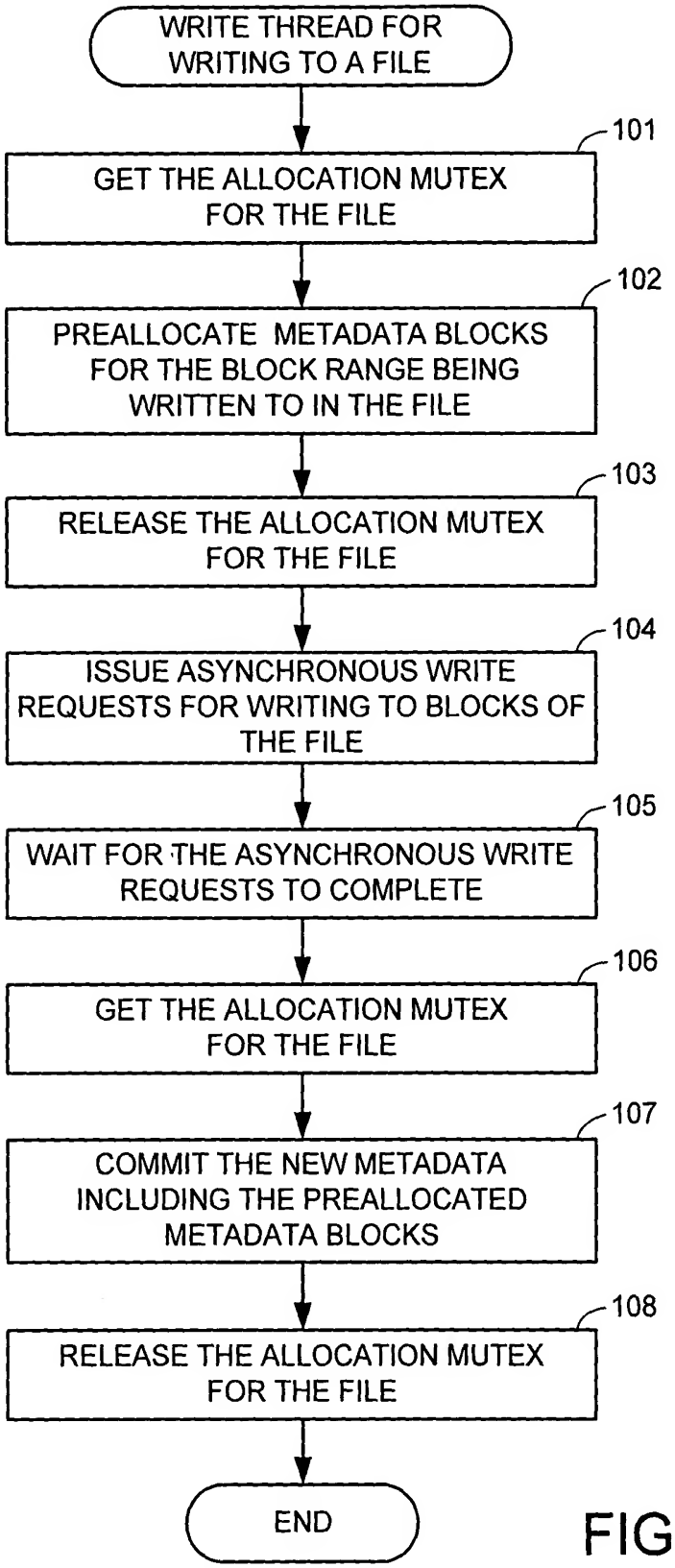

WRITE THREAD FOR WRITING TO A FILE
101
GET THE ALLOCATION MUTEX FOR THE FILE
102
PREALLOCATE METADATA BLOCKS FOR THE BLOCK RANGE BEING WRITTEN TO IN THE FILE
103
RELEASE THE ALLOCATION MUTEX FOR THE FILE
104
ISSUE ASYNCHRONOUS WRITE REQUESTS FOR WRITING TO BLOCKS OF THE FILE
105
WAIT FOR THE ASYNCHRONOUS WRITE REQUESTS TO COMPLETE
106
GET THE ALLOCATION MUTEX FOR THE FILE
107
COMMIT THE NEW METADATA INCLUDING THE PREALLOCATED METADATA BLOCKS
108
RELEASE THE ALLOCATION MUTEX FOR THE FILE
END

FIG. 11

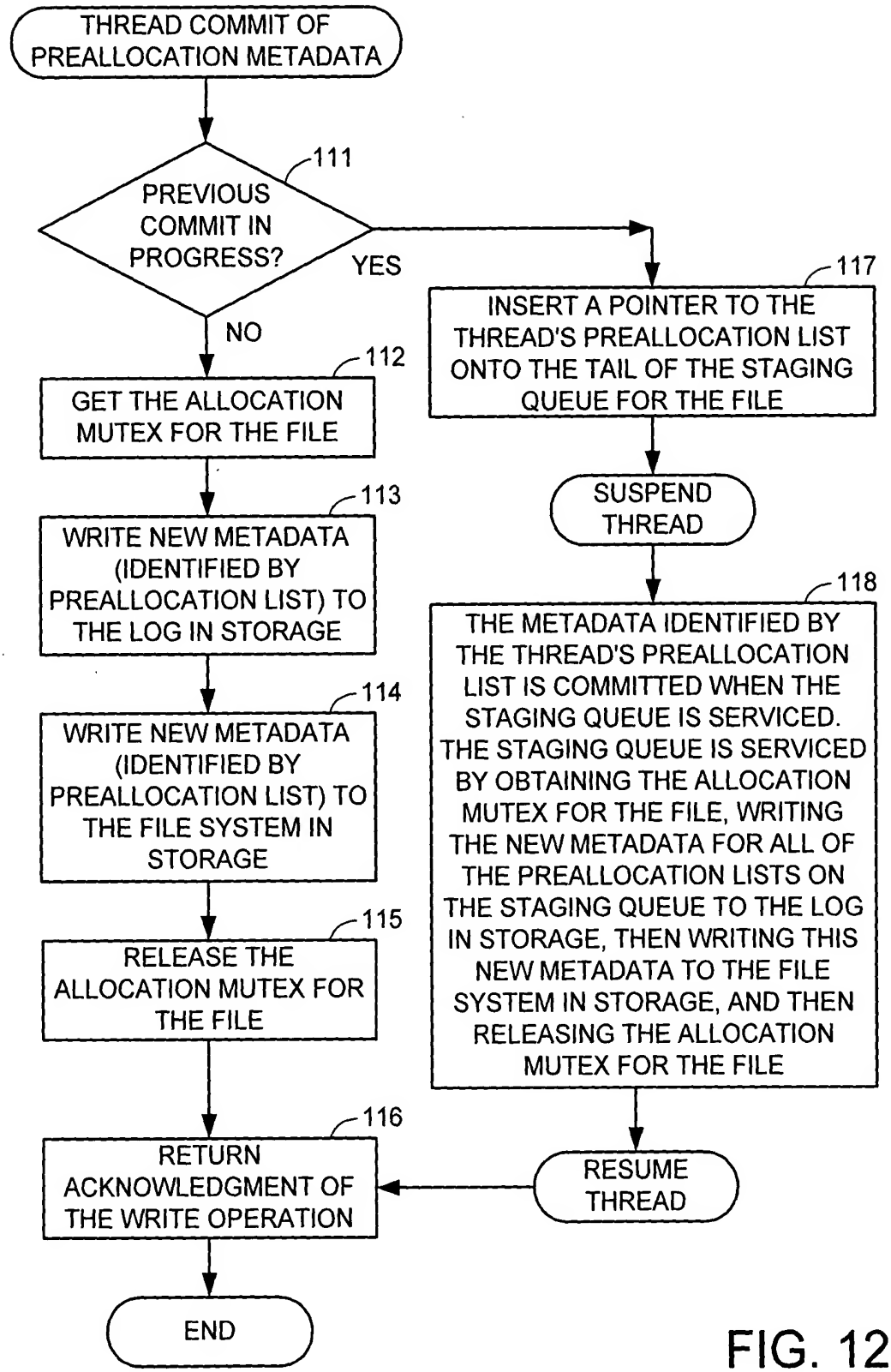

THREAD COMMIT OF PREALLOCATION METADATA
111
PREVIOUS COMMIT IN PROGRESS?
YES
NO
112
GET THE ALLOCATION MUTEX FOR THE FILE
113
WRITE NEW METADATA (IDENTIFIED BY PREALLOCATION LIST) TO THE LOG IN STORAGE
114
WRITE NEW METADATA (IDENTIFIED BY PREALLOCATION LIST) TO THE FILE SYSTEM IN STORAGE
115
RELEASE THE ALLOCATION MUTEX FOR THE FILE
116
RETURN ACKNOWLEDGMENT OF THE WRITE OPERATION
END
117
INSERT A POINTER TO THE THREAD'S PREALLOCATION LIST ONTO THE TAIL OF THE STAGING QUEUE FOR THE FILE
SUSPEND THREAD
118
THE METADATA IDENTIFIED BY THE THREAD'S PREALLOCATION LIST IS COMMITTED WHEN THE STAGING QUEUE IS SERVICED. THE STAGING QUEUE IS SERVICED BY OBTAINING THE ALLOCATION MUTEX FOR THE FILE, WRITING THE NEW METADATA FOR ALL OF THE PREALLOCATION LISTS ON THE STAGING QUEUE TO THE LOG IN STORAGE, THEN WRITING THIS NEW METADATA TO THE FILE SYSTEM IN STORAGE, AND THEN RELEASING THE ALLOCATION MUTEX FOR THE FILE
RESUME THREAD

FIG. 12

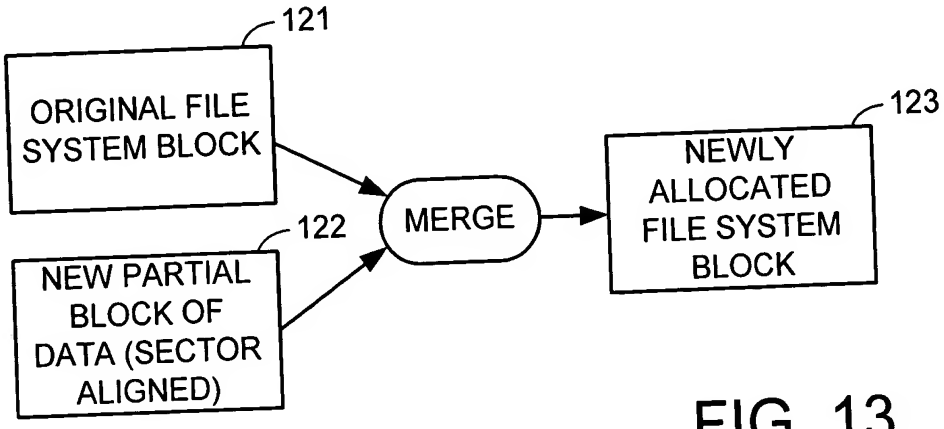

121
ORIGINAL FILE SYSTEM BLOCK
122
NEW PARTIAL BLOCK OF DATA (SECTOR ALIGNED)
MERGE
123
NEWLY ALLOCATED FILE SYSTEM BLOCK

FIG. 13

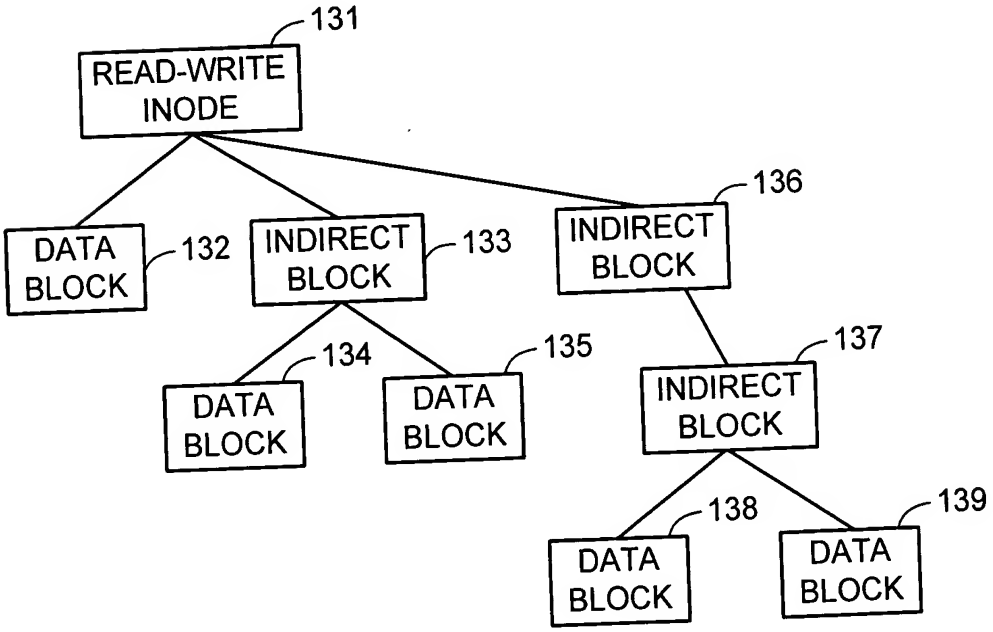

131
READ-WRITE INODE
DATA BLOCK
132
INDIRECT BLOCK
133
INDIRECT BLOCK
136
DATA BLOCK
134
DATA BLOCK
135
INDIRECT BLOCK
137
DATA BLOCK
138
DATA BLOCK
139

FIG. 14

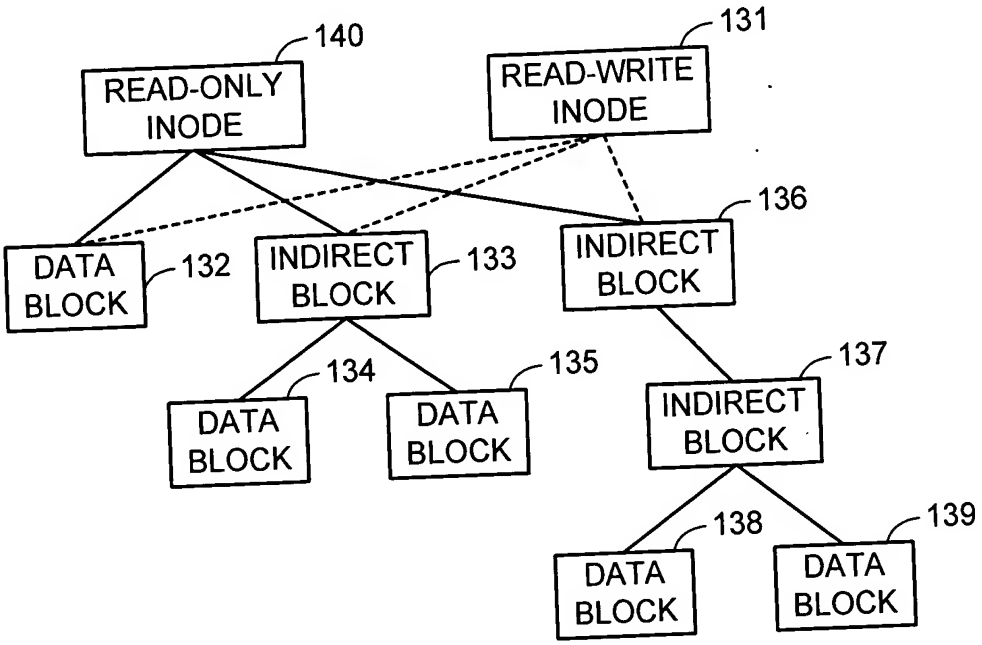

140
READ-ONLY INODE
131
READ-WRITE INODE
136
DATA BLOCK
132
INDIRECT BLOCK
133
INDIRECT BLOCK
134
DATA BLOCK
135
DATA BLOCK
137
INDIRECT BLOCK
138
DATA BLOCK
139
DATA BLOCK

FIG. 15

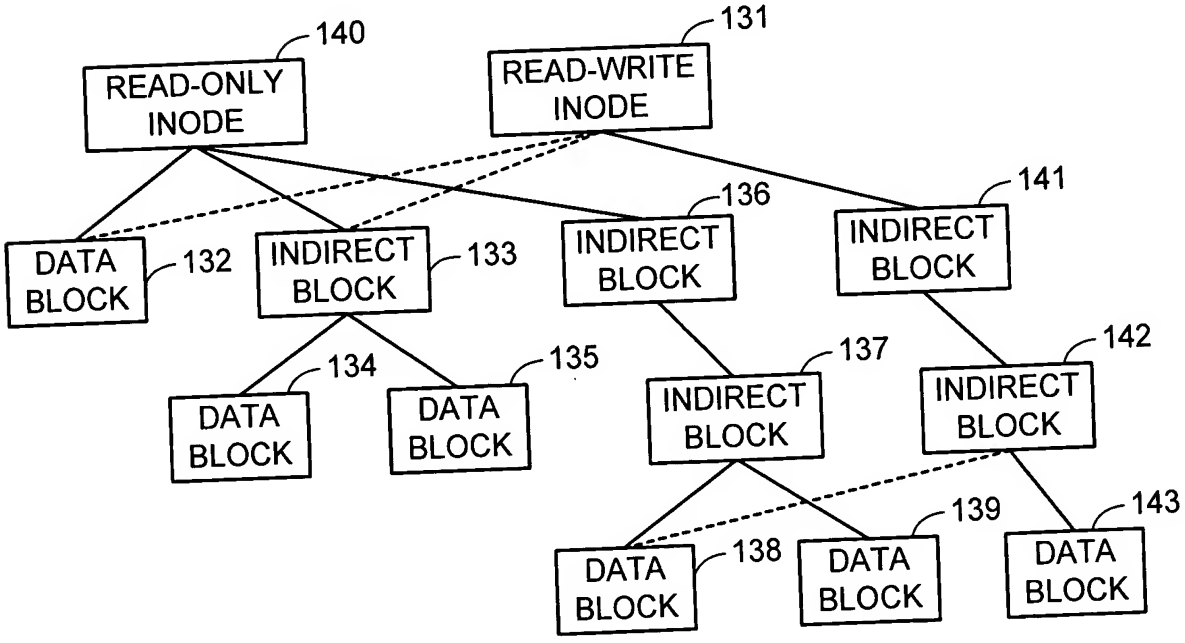

140
READ-ONLY INODE
131
READ-WRITE INODE
136
141
DATA BLOCK
132
INDIRECT BLOCK
133
INDIRECT BLOCK
INDIRECT BLOCK
134
DATA BLOCK
135
DATA BLOCK
137
INDIRECT BLOCK
142
INDIRECT BLOCK
139
143
DATA BLOCK
138
DATA BLOCK
DATA BLOCK

FIG. 16

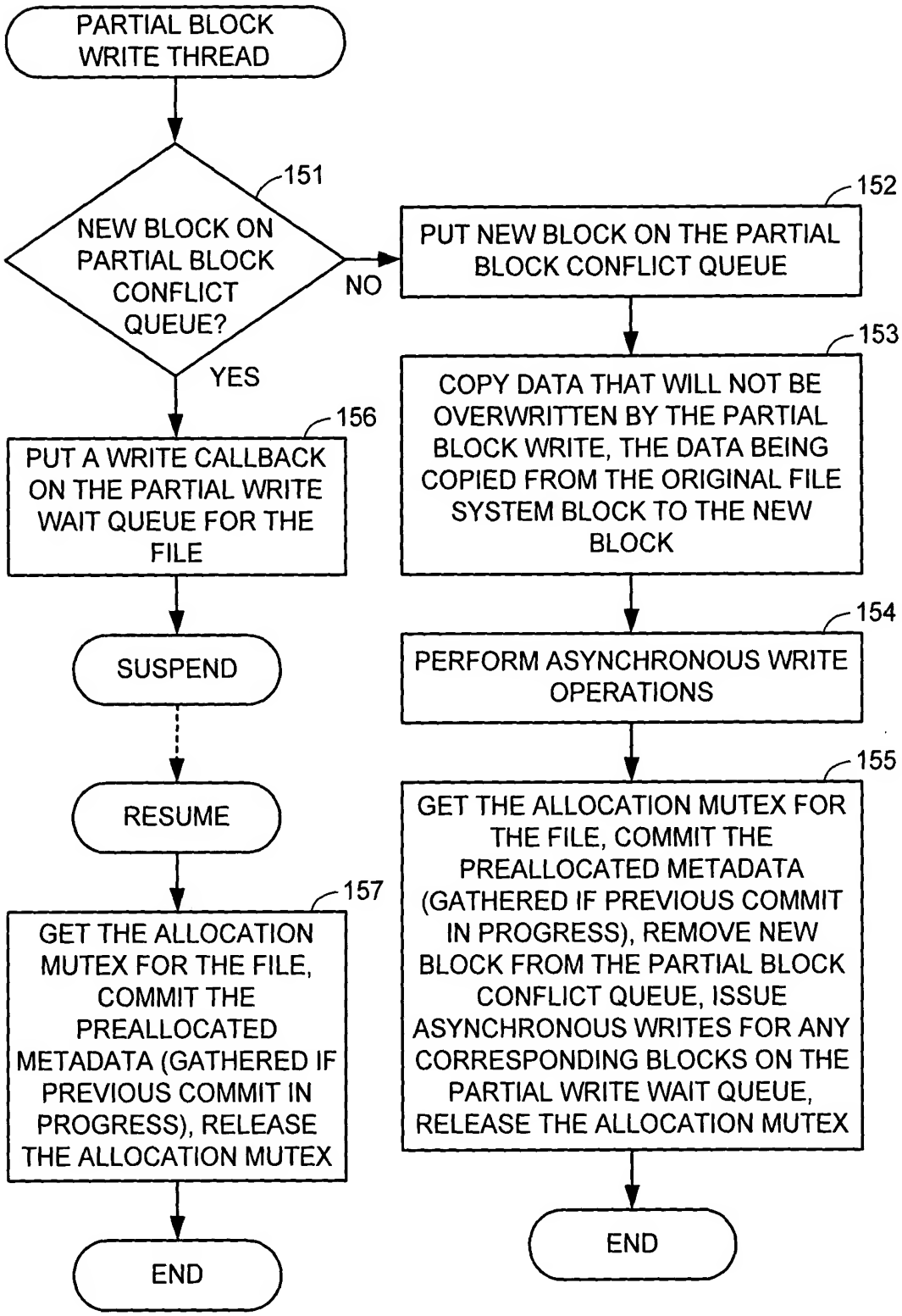

PARTIAL BLOCK WRITE THREAD
151
NEW BLOCK ON PARTIAL BLOCK CONFLICT QUEUE?
NO
YES
152
PUT NEW BLOCK ON THE PARTIAL BLOCK CONFLICT QUEUE
153
COPY DATA THAT WILL NOT BE OVERWRITTEN BY THE PARTIAL BLOCK WRITE, THE DATA BEING COPIED FROM THE ORIGINAL FILE SYSTEM BLOCK TO THE NEW BLOCK
154
PERFORM ASYNCHRONOUS WRITE OPERATIONS
155
GET THE ALLOCATION MUTEX FOR THE FILE, COMMIT THE PREALLOCATED METADATA (GATHERED IF PREVIOUS COMMIT IN PROGRESS), REMOVE NEW BLOCK FROM THE PARTIAL BLOCK CONFLICT QUEUE, ISSUE ASYNCHRONOUS WRITES FOR ANY CORRESPONDING BLOCKS ON THE PARTIAL WRITE WAIT QUEUE, RELEASE THE ALLOCATION MUTEX
END
156
PUT A WRITE CALLBACK ON THE PARTIAL WRITE WAIT QUEUE FOR THE FILE
SUSPEND
RESUME
157
GET THE ALLOCATION MUTEX FOR THE FILE, COMMIT THE PREALLOCATED METADATA (GATHERED IF PREVIOUS COMMIT IN PROGRESS), RELEASE THE ALLOCATION MUTEX
END

FIG. 17

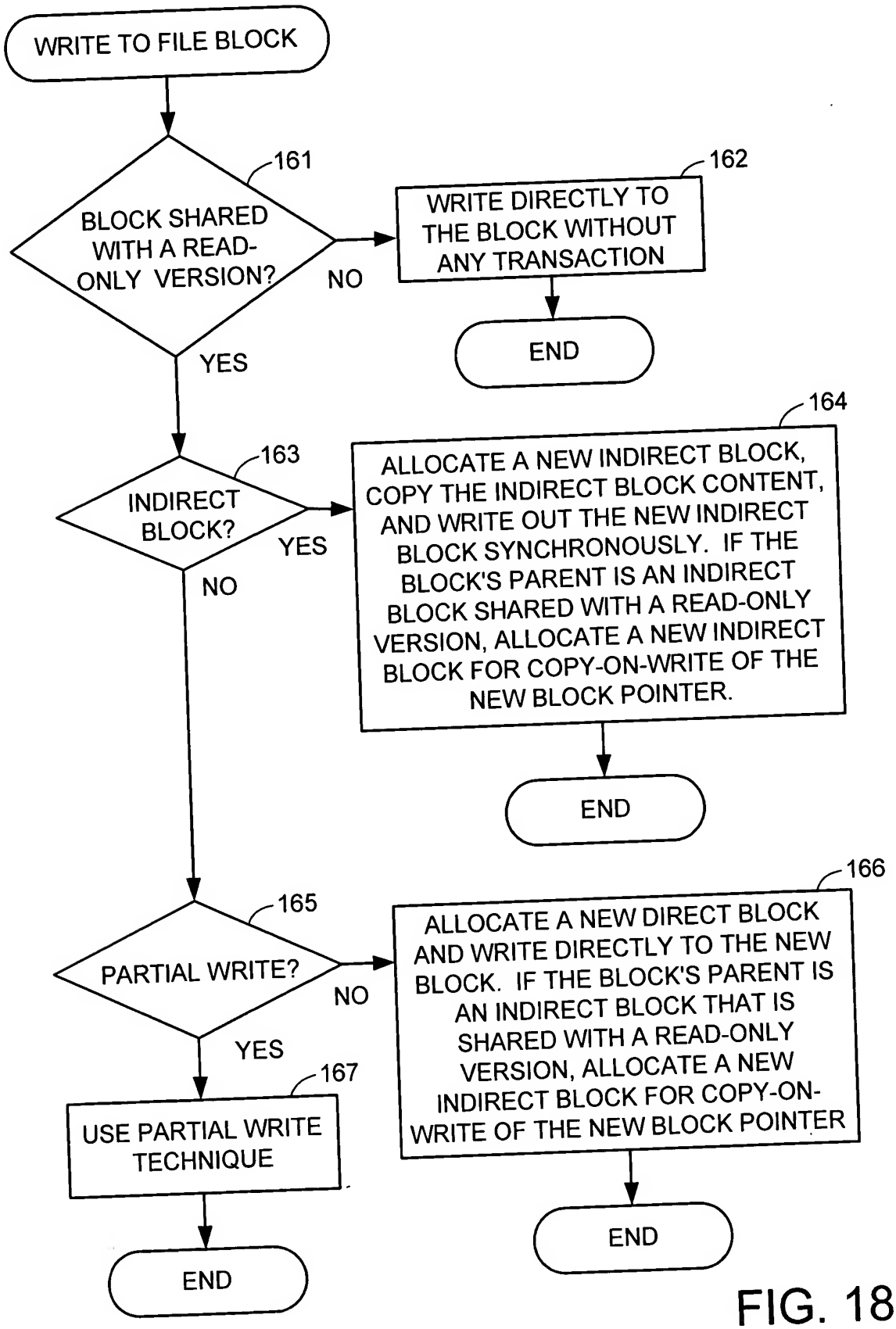

WRITE TO FILE BLOCK
161
BLOCK SHARED WITH A READ-ONLY VERSION?
NO
162
WRITE DIRECTLY TO THE BLOCK WITHOUT ANY TRANSACTION
END
YES
163
INDIRECT BLOCK?
YES
164
ALLOCATE A NEW INDIRECT BLOCK, COPY THE INDIRECT BLOCK CONTENT, AND WRITE OUT THE NEW INDIRECT BLOCK SYNCHRONOUSLY. IF THE BLOCK'S PARENT IS AN INDIRECT BLOCK SHARED WITH A READ-ONLY VERSION, ALLOCATE A NEW INDIRECT BLOCK FOR COPY-ON-WRITE OF THE NEW BLOCK POINTER.
END
NO
165
PARTIAL WRITE?
NO
166
ALLOCATE A NEW DIRECT BLOCK AND WRITE DIRECTLY TO THE NEW BLOCK. IF THE BLOCK'S PARENT IS AN INDIRECT BLOCK THAT IS SHARED WITH A READ-ONLY VERSION, ALLOCATE A NEW INDIRECT BLOCK FOR COPY-ON-WRITE OF THE NEW BLOCK POINTER
END
YES
167
USE PARTIAL WRITE TECHNIQUE
END

FIG. 18

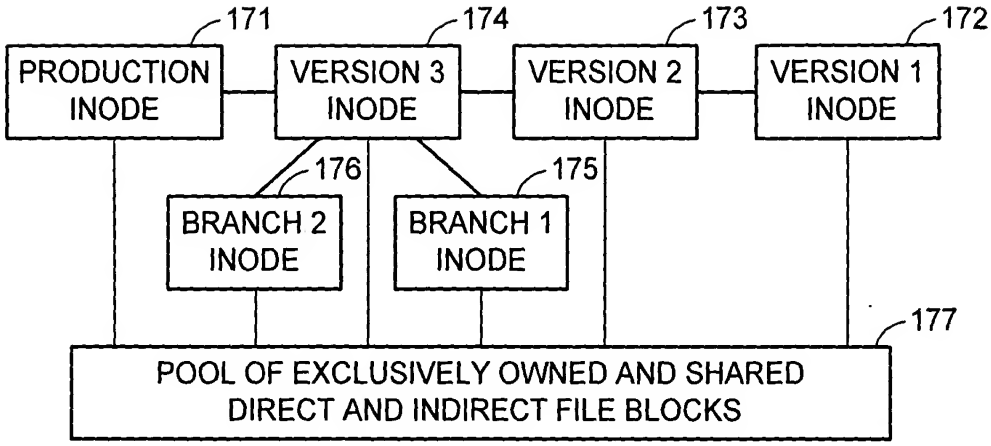

171
174
173
172
PRODUCTION INODE
VERSION 3 INODE
VERSION 2 INODE
VERSION 1 INODE
176
175
BRANCH 2 INODE
BRANCH 1 INODE
177
POOL OF EXCLUSIVELY OWNED AND SHARED DIRECT AND INDIRECT FILE BLOCKS

FIG. 19

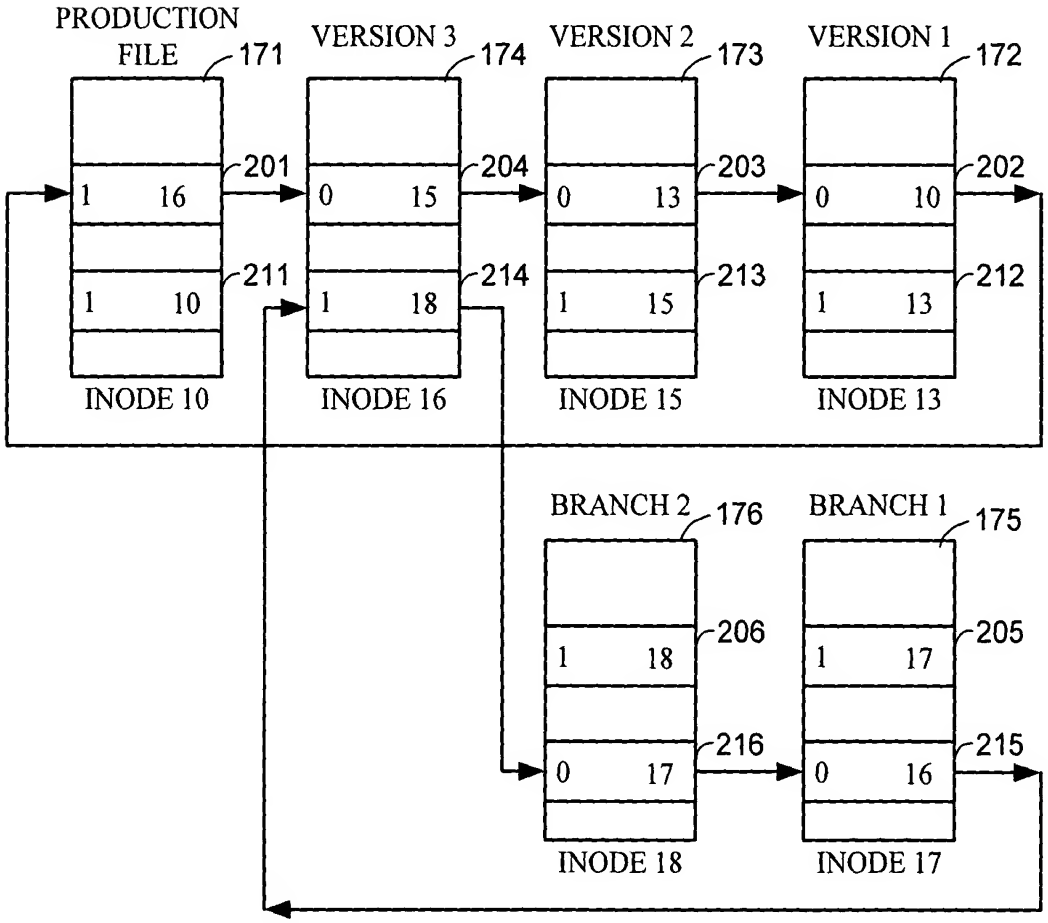

PRODUCTION FILE
171
VERSION 3
174
VERSION 2
173
VERSION 1
172
201
1 16
211
1 10
204
0 15
214
1 18
203
0 13
213
1 15
202
0 10
212
1 13
INODE 10
INODE 16
INODE 15
INODE 13
BRANCH 2
176
BRANCH 1
175
206
1 18
205
1 17
216
0 17
215
0 16
INODE 18
INODE 17

FIG. 23

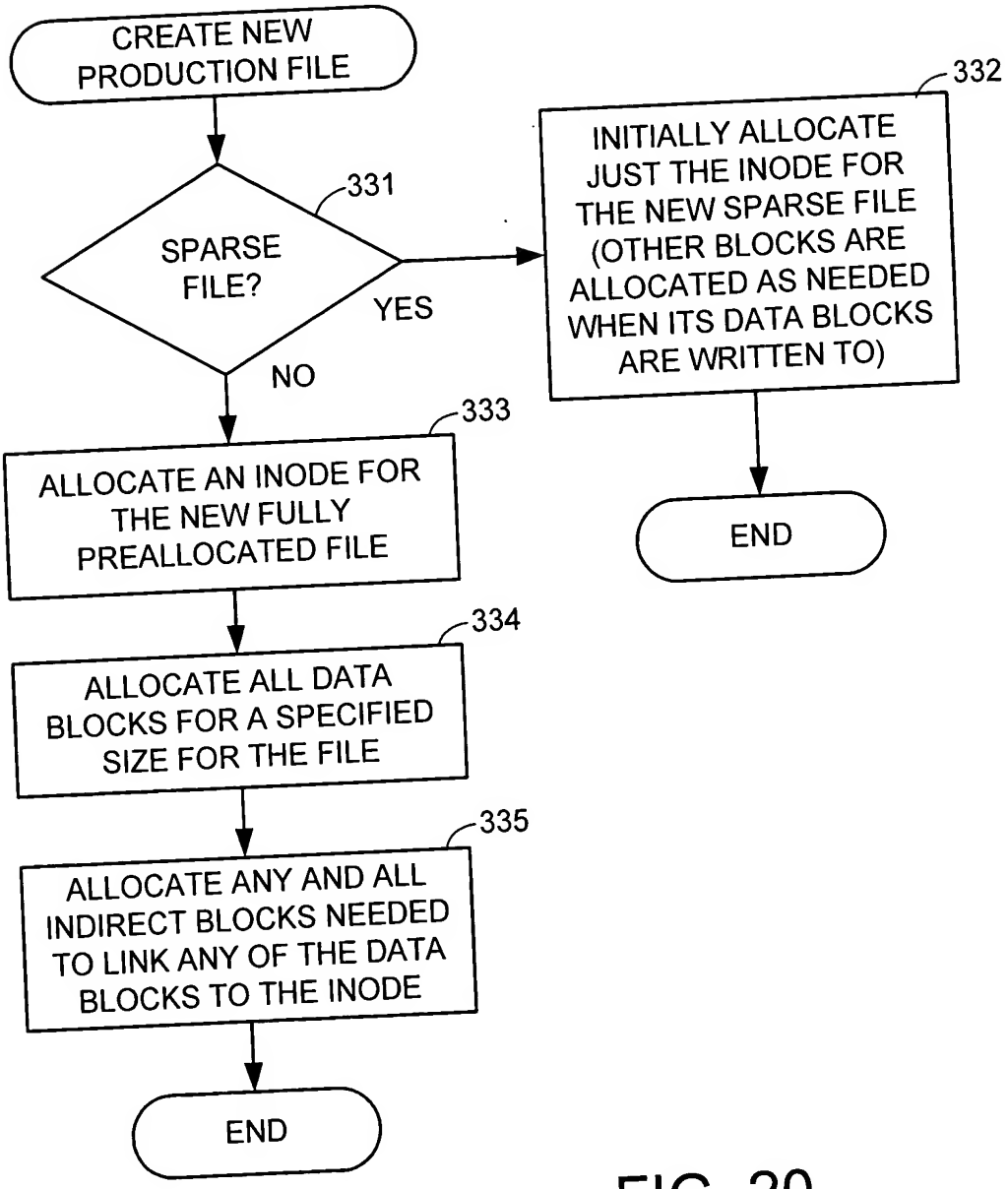

CREATE NEW PRODUCTION FILE
331
SPARSE FILE?
YES
NO
332
INITIALLY ALLOCATE JUST THE INODE FOR THE NEW SPARSE FILE (OTHER BLOCKS ARE ALLOCATED AS NEEDED WHEN ITS DATA BLOCKS ARE WRITTEN TO)
END
333
ALLOCATE AN INODE FOR THE NEW FULLY PREALLOCATED FILE
334
ALLOCATE ALL DATA BLOCKS FOR A SPECIFIED SIZE FOR THE FILE
335
ALLOCATE ANY AND ALL INDIRECT BLOCKS NEEDED TO LINK ANY OF THE DATA BLOCKS TO THE INODE
END

FIG. 20

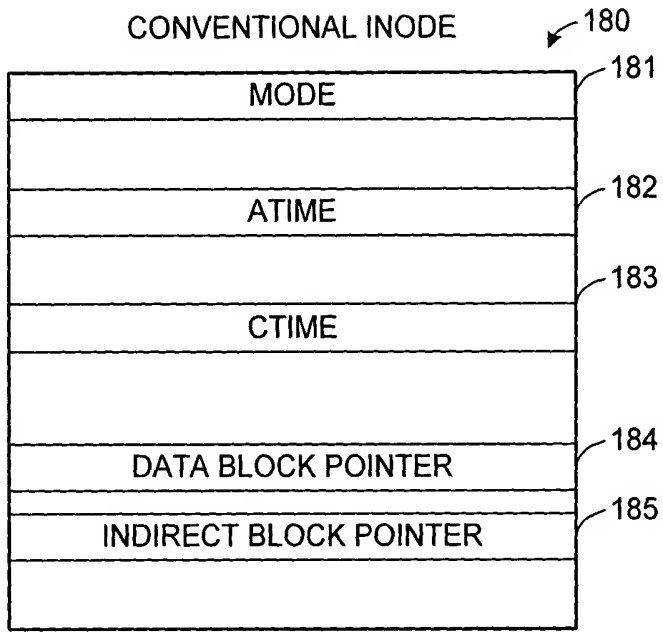


FIG. 21 (PRIOR ART)

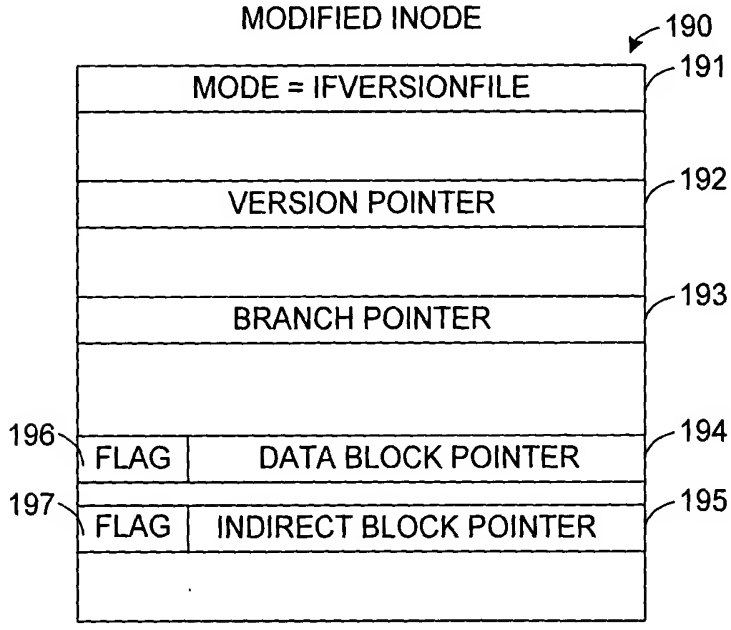


FIG. 22

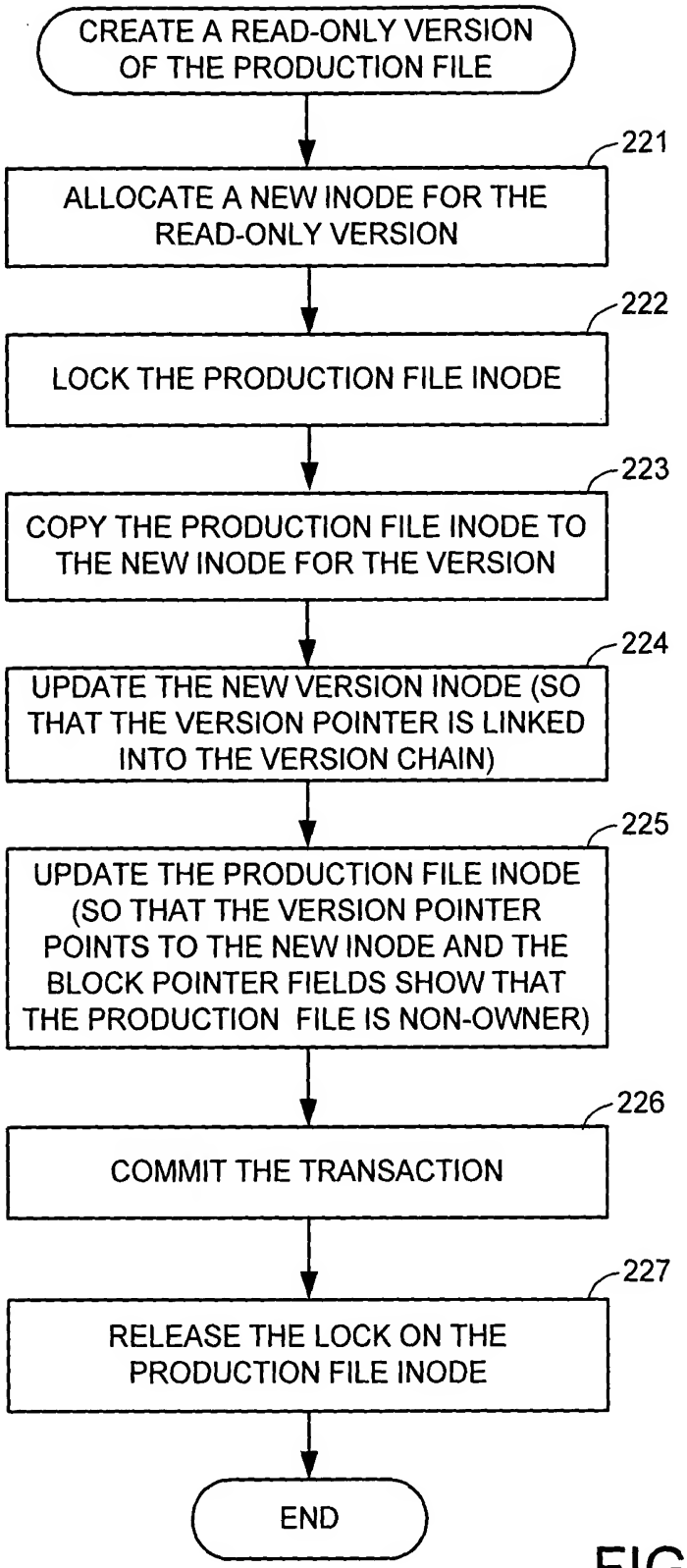

CREATE A READ-ONLY VERSION OF THE PRODUCTION FILE
221
ALLOCATE A NEW INODE FOR THE READ-ONLY VERSION
222
LOCK THE PRODUCTION FILE INODE
223
COPY THE PRODUCTION FILE INODE TO THE NEW INODE FOR THE VERSION
224
UPDATE THE NEW VERSION INODE (SO THAT THE VERSION POINTER IS LINKED INTO THE VERSION CHAIN)
225
UPDATE THE PRODUCTION FILE INODE (SO THAT THE VERSION POINTER POINTS TO THE NEW INODE AND THE BLOCK POINTER FIELDS SHOW THAT THE PRODUCTION FILE IS NON-OWNER)
226
COMMIT THE TRANSACTION
227
RELEASE THE LOCK ON THE PRODUCTION FILE INODE
END

FIG. 24

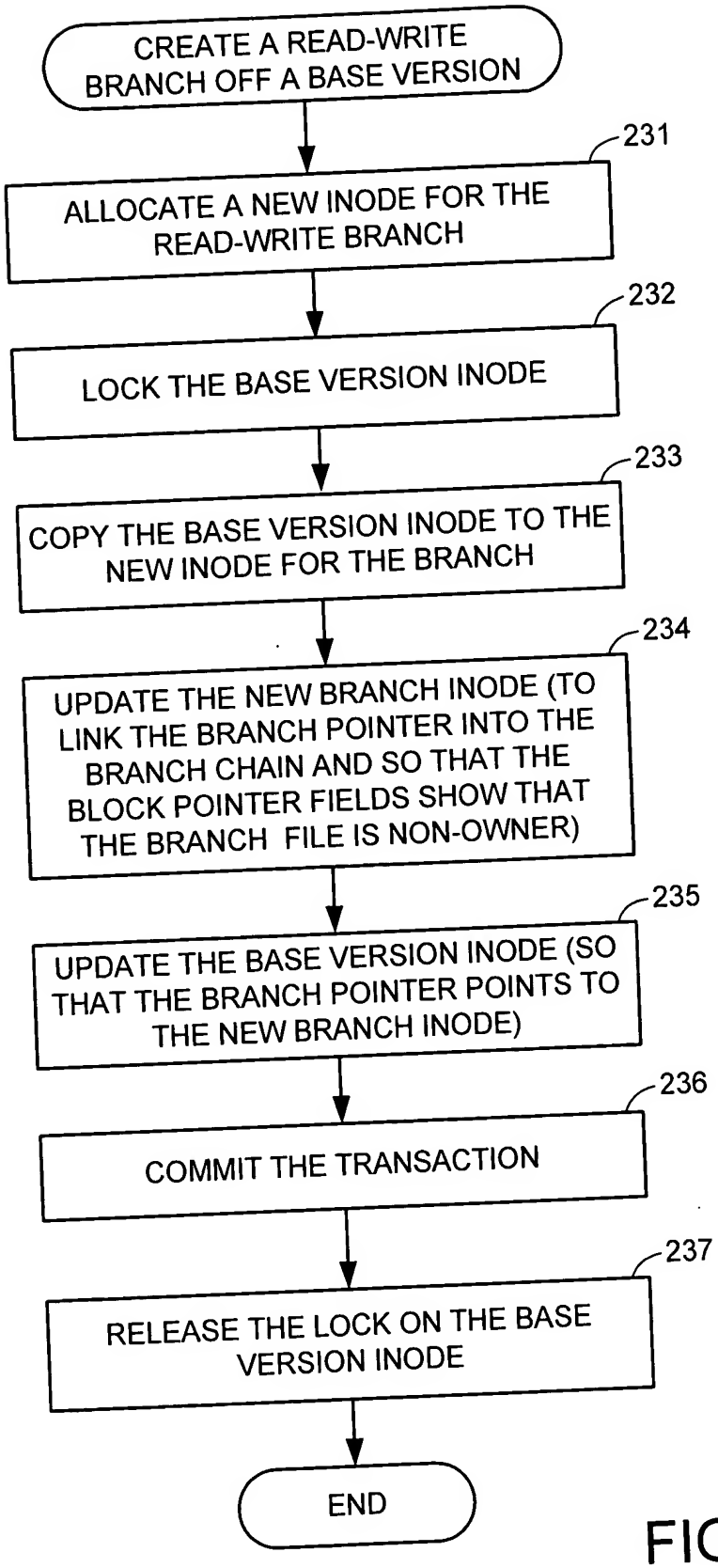
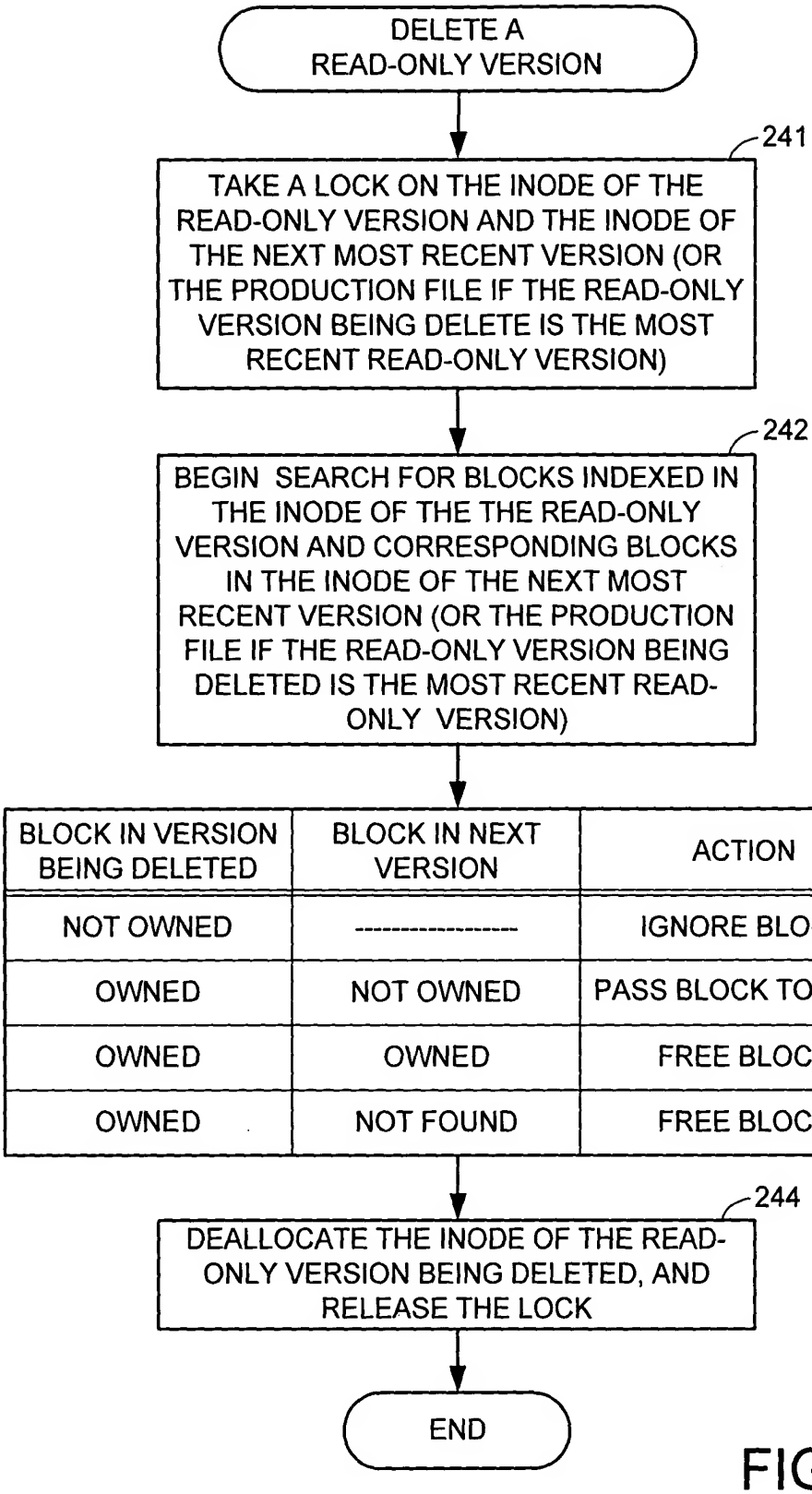

CREATE A READ-WRITE BRANCH OFF A BASE VERSION
231
ALLOCATE A NEW INODE FOR THE READ-WRITE BRANCH
232
LOCK THE BASE VERSION INODE
233
COPY THE BASE VERSION INODE TO THE NEW INODE FOR THE BRANCH
234
UPDATE THE NEW BRANCH INODE (TO LINK THE BRANCH POINTER INTO THE BRANCH CHAIN AND SO THAT THE BLOCK POINTER FIELDS SHOW THAT THE BRANCH FILE IS NON-OWNER)
235
UPDATE THE BASE VERSION INODE (SO THAT THE BRANCH POINTER POINTS TO THE NEW BRANCH INODE)
236
COMMIT THE TRANSACTION
237
RELEASE THE LOCK ON THE BASE VERSION INODE
END

FIG. 25


DELETE A READ-ONLY VERSION
241
TAKE A LOCK ON THE INODE OF THE READ-ONLY VERSION AND THE INODE OF THE NEXT MOST RECENT VERSION (OR THE PRODUCTION FILE IF THE READ-ONLY VERSION BEING DELETE IS THE MOST RECENT READ-ONLY VERSION)
242
BEGIN SEARCH FOR BLOCKS INDEXED IN THE INODE OF THE THE READ-ONLY VERSION AND CORRESPONDING BLOCKS IN THE INODE OF THE NEXT MOST RECENT VERSION (OR THE PRODUCTION FILE IF THE READ-ONLY VERSION BEING DELETED IS THE MOST RECENT READ-ONLY VERSION)
243
BLOCK IN VERSION BEING DELETED
BLOCK IN NEXT VERSION
ACTION
NOT OWNED

IGNORE BLOCK
OWNED
NOT OWNED
PASS BLOCK TO NEXT
OWNED
OWNED
FREE BLOCK
OWNED
NOT FOUND
FREE BLOCK
244
DEALLOCATE THE INODE OF THE READ-ONLY VERSION BEING DELETED, AND RELEASE THE LOCK
END

FIG. 26

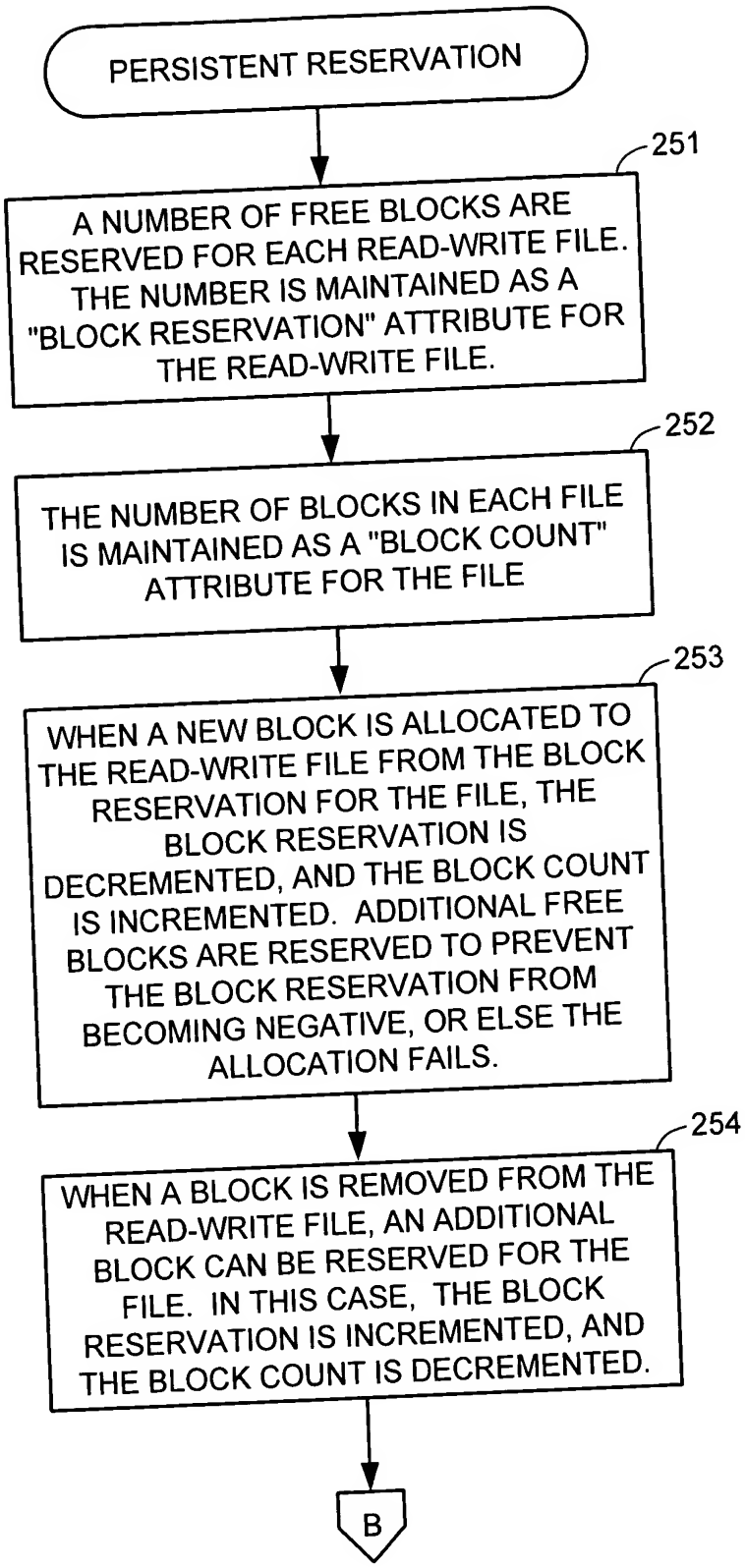

PERSISTENT RESERVATION
251
A NUMBER OF FREE BLOCKS ARE RESERVED FOR EACH READ-WRITE FILE. THE NUMBER IS MAINTAINED AS A "BLOCK RESERVATION" ATTRIBUTE FOR THE READ-WRITE FILE.
252
THE NUMBER OF BLOCKS IN EACH FILE IS MAINTAINED AS A "BLOCK COUNT" ATTRIBUTE FOR THE FILE
253
WHEN A NEW BLOCK IS ALLOCATED TO THE READ-WRITE FILE FROM THE BLOCK RESERVATION FOR THE FILE, THE BLOCK RESERVATION IS DECREMENTED, AND THE BLOCK COUNT IS INCREMENTED. ADDITIONAL FREE BLOCKS ARE RESERVED TO PREVENT THE BLOCK RESERVATION FROM BECOMING NEGATIVE, OR ELSE THE ALLOCATION FAILS.
254
WHEN A BLOCK IS REMOVED FROM THE READ-WRITE FILE, AN ADDITIONAL BLOCK CAN BE RESERVED FOR THE FILE. IN THIS CASE, THE BLOCK RESERVATION IS INCREMENTED, AND THE BLOCK COUNT IS DECREMENTED.
B

FIG. 27

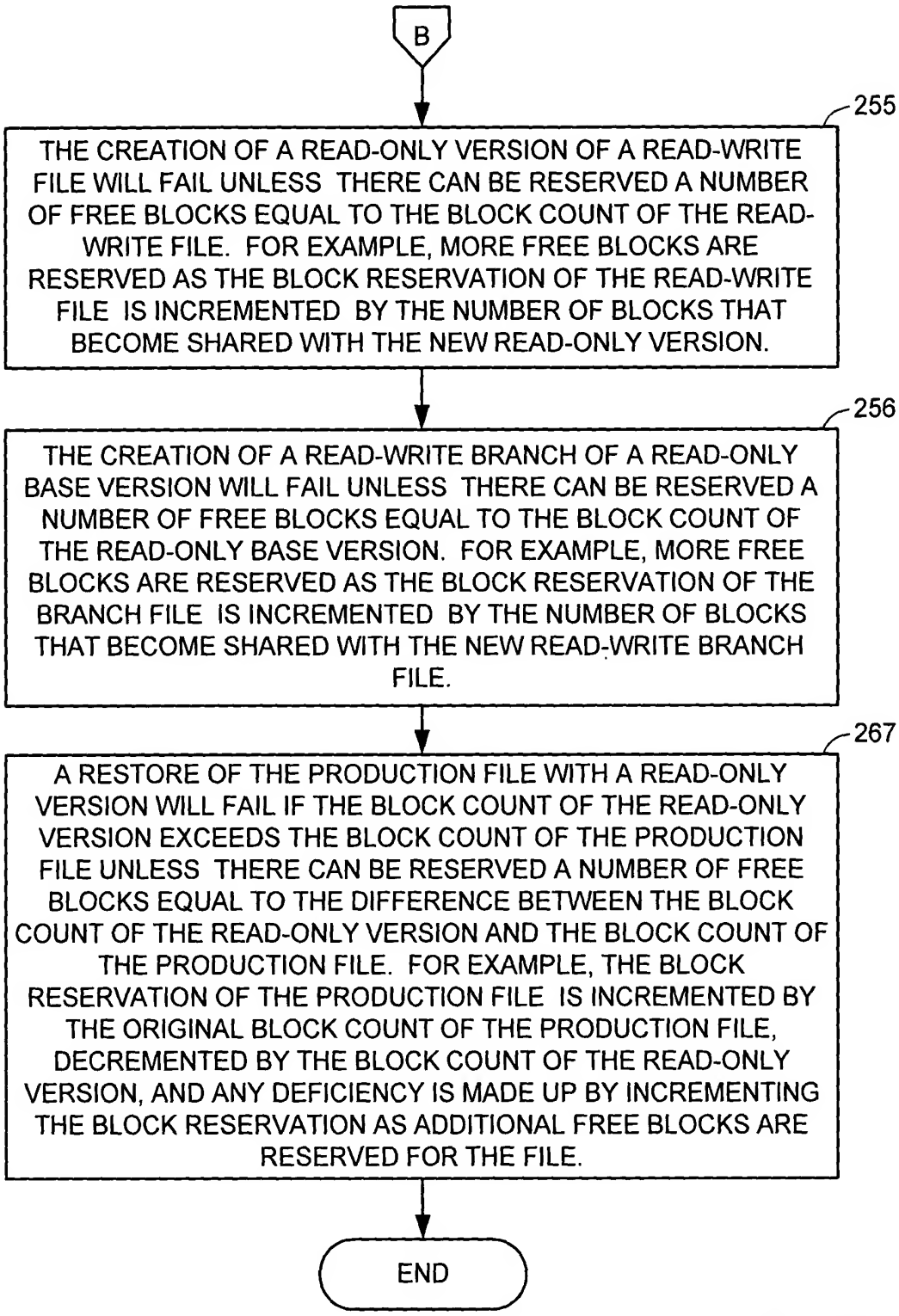

B
255
THE CREATION OF A READ-ONLY VERSION OF A READ-WRITE FILE WILL FAIL UNLESS THERE CAN BE RESERVED A NUMBER OF FREE BLOCKS EQUAL TO THE BLOCK COUNT OF THE READ-WRITE FILE. FOR EXAMPLE, MORE FREE BLOCKS ARE RESERVED AS THE BLOCK RESERVATION OF THE READ-WRITE FILE IS INCREMENTED BY THE NUMBER OF BLOCKS THAT BECOME SHARED WITH THE NEW READ-ONLY VERSION.
256
THE CREATION OF A READ-WRITE BRANCH OF A READ-ONLY BASE VERSION WILL FAIL UNLESS THERE CAN BE RESERVED A NUMBER OF FREE BLOCKS EQUAL TO THE BLOCK COUNT OF THE READ-ONLY BASE VERSION. FOR EXAMPLE, MORE FREE BLOCKS ARE RESERVED AS THE BLOCK RESERVATION OF THE BRANCH FILE IS INCREMENTED BY THE NUMBER OF BLOCKS THAT BECOME SHARED WITH THE NEW READ-WRITE BRANCH FILE.
267
A RESTORE OF THE PRODUCTION FILE WITH A READ-ONLY VERSION WILL FAIL IF THE BLOCK COUNT OF THE READ-ONLY VERSION EXCEEDS THE BLOCK COUNT OF THE PRODUCTION FILE UNLESS THERE CAN BE RESERVED A NUMBER OF FREE BLOCKS EQUAL TO THE DIFFERENCE BETWEEN THE BLOCK COUNT OF THE READ-ONLY VERSION AND THE BLOCK COUNT OF THE PRODUCTION FILE. FOR EXAMPLE, THE BLOCK RESERVATION OF THE PRODUCTION FILE IS INCREMENTED BY THE ORIGINAL BLOCK COUNT OF THE PRODUCTION FILE, DECREMENTED BY THE BLOCK COUNT OF THE READ-ONLY VERSION, AND ANY DEFICIENCY IS MADE UP BY INCREMENTING THE BLOCK RESERVATION AS ADDITIONAL FREE BLOCKS ARE RESERVED FOR THE FILE.
END

FIG. 28

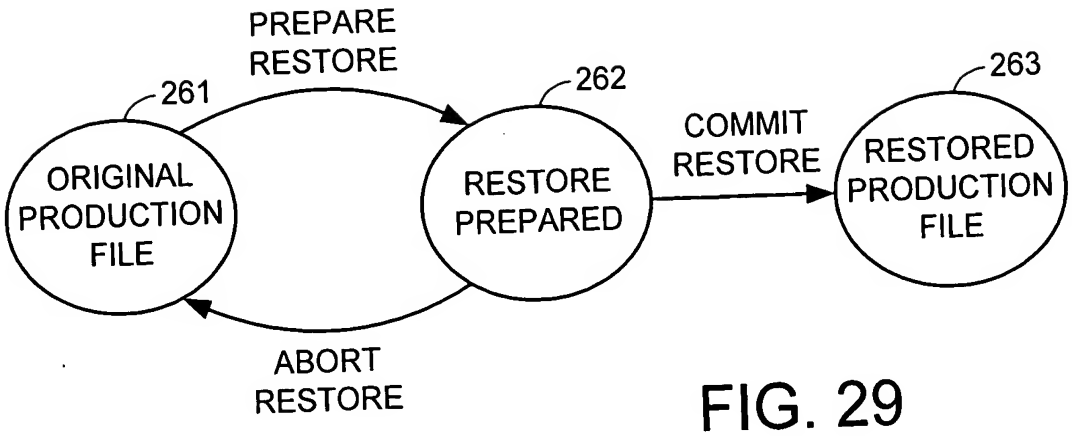

PREPARE
RESTORE
261
262
263
COMMIT
RESTORE
ORIGINAL
PRODUCTION
FILE
RESTORE
PREPARED
RESTORED
PRODUCTION
FILE
ABORT
RESTORE

FIG. 29

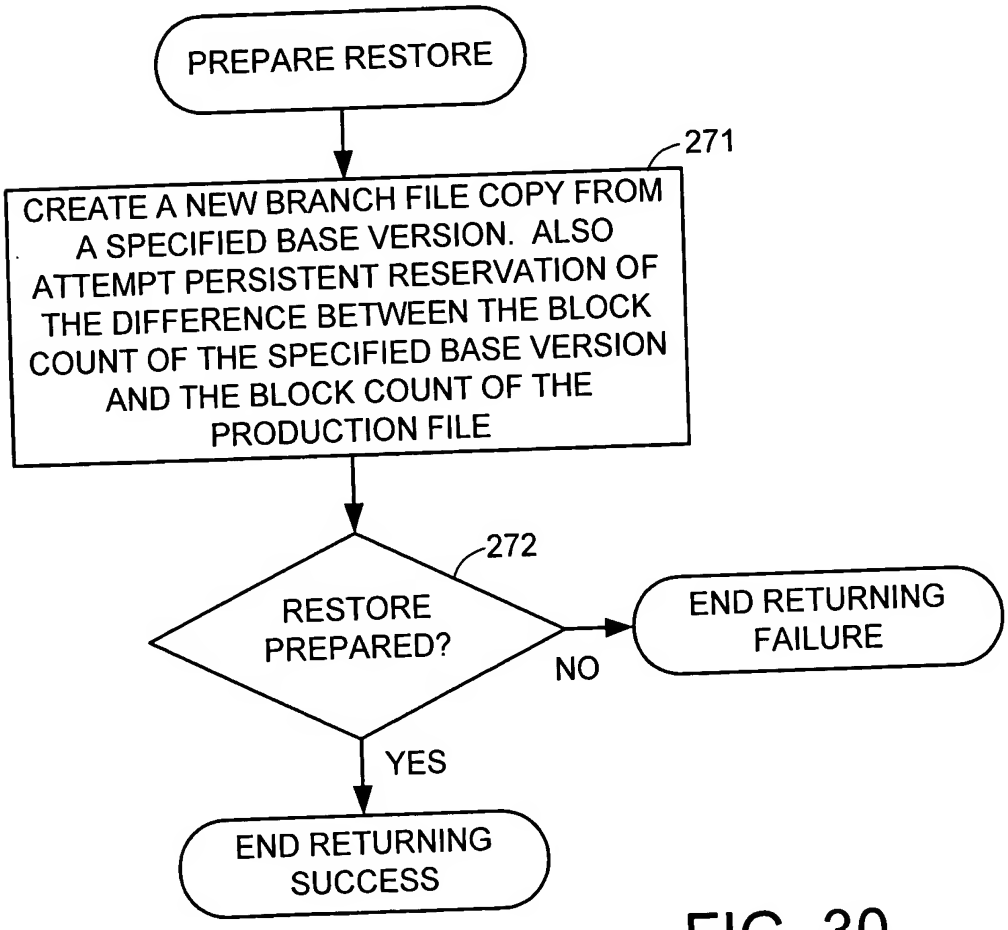

PREPARE RESTORE
271
CREATE A NEW BRANCH FILE COPY FROM A SPECIFIED BASE VERSION. ALSO ATTEMPT PERSISTENT RESERVATION OF THE DIFFERENCE BETWEEN THE BLOCK COUNT OF THE SPECIFIED BASE VERSION AND THE BLOCK COUNT OF THE PRODUCTION FILE
272
RESTORE PREPARED?
NO
END RETURNING FAILURE
YES
END RETURNING SUCCESS

FIG. 30

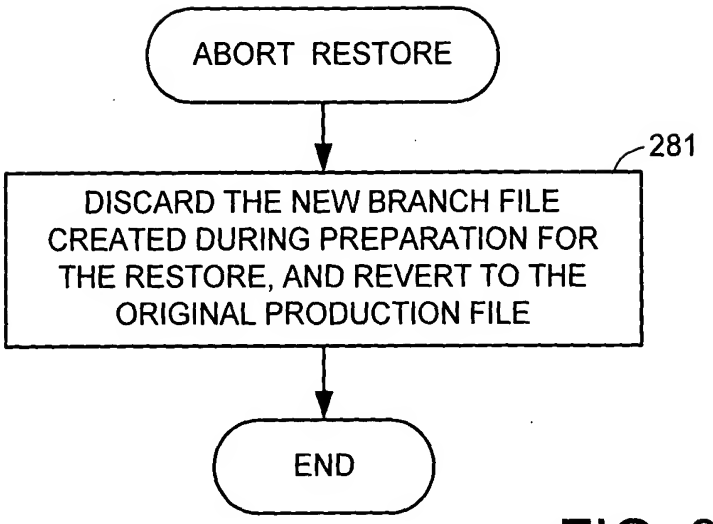

ABORT RESTORE
281
DISCARD THE NEW BRANCH FILE CREATED DURING PREPARATION FOR THE RESTORE, AND REVERT TO THE ORIGINAL PRODUCTION FILE
END

FIG. 31

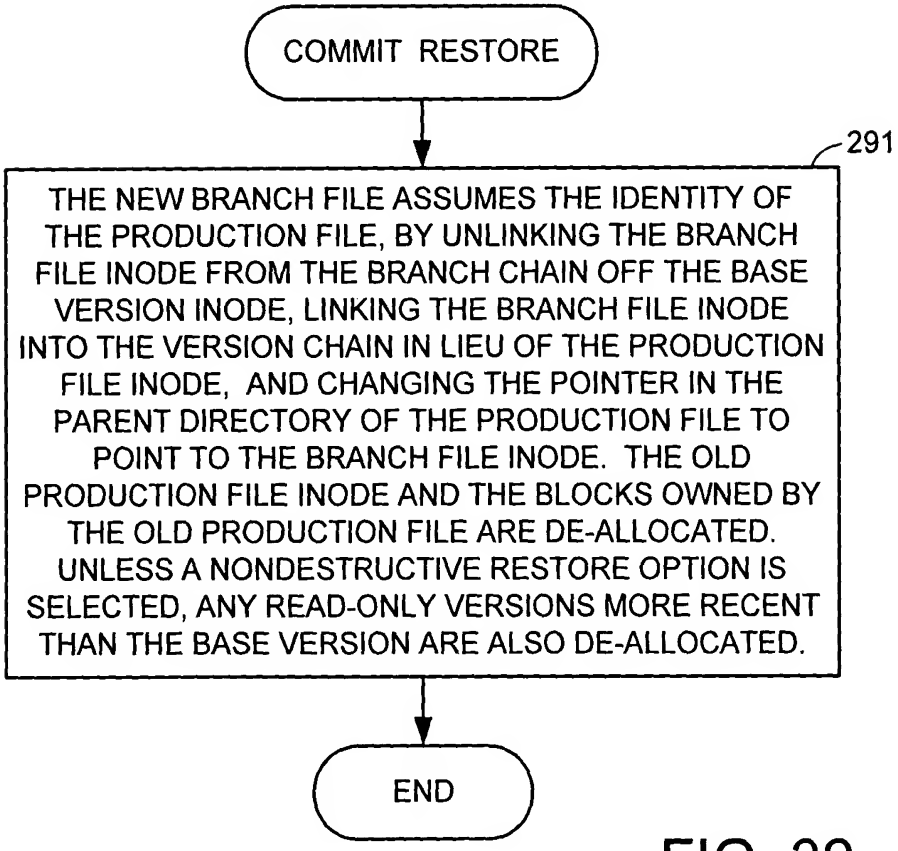

COMMIT RESTORE
291
THE NEW BRANCH FILE ASSUMES THE IDENTITY OF THE PRODUCTION FILE, BY UNLINKING THE BRANCH FILE INODE FROM THE BRANCH CHAIN OFF THE BASE VERSION INODE, LINKING THE BRANCH FILE INODE INTO THE VERSION CHAIN IN LIEU OF THE PRODUCTION FILE INODE, AND CHANGING THE POINTER IN THE PARENT DIRECTORY OF THE PRODUCTION FILE TO POINT TO THE BRANCH FILE INODE. THE OLD PRODUCTION FILE INODE AND THE BLOCKS OWNED BY THE OLD PRODUCTION FILE ARE DE-ALLOCATED. UNLESS A NONDESTRUCTIVE RESTORE OPTION IS SELECTED, ANY READ-ONLY VERSIONS MORE RECENT THAN THE BASE VERSION ARE ALSO DE-ALLOCATED.
END

FIG. 32

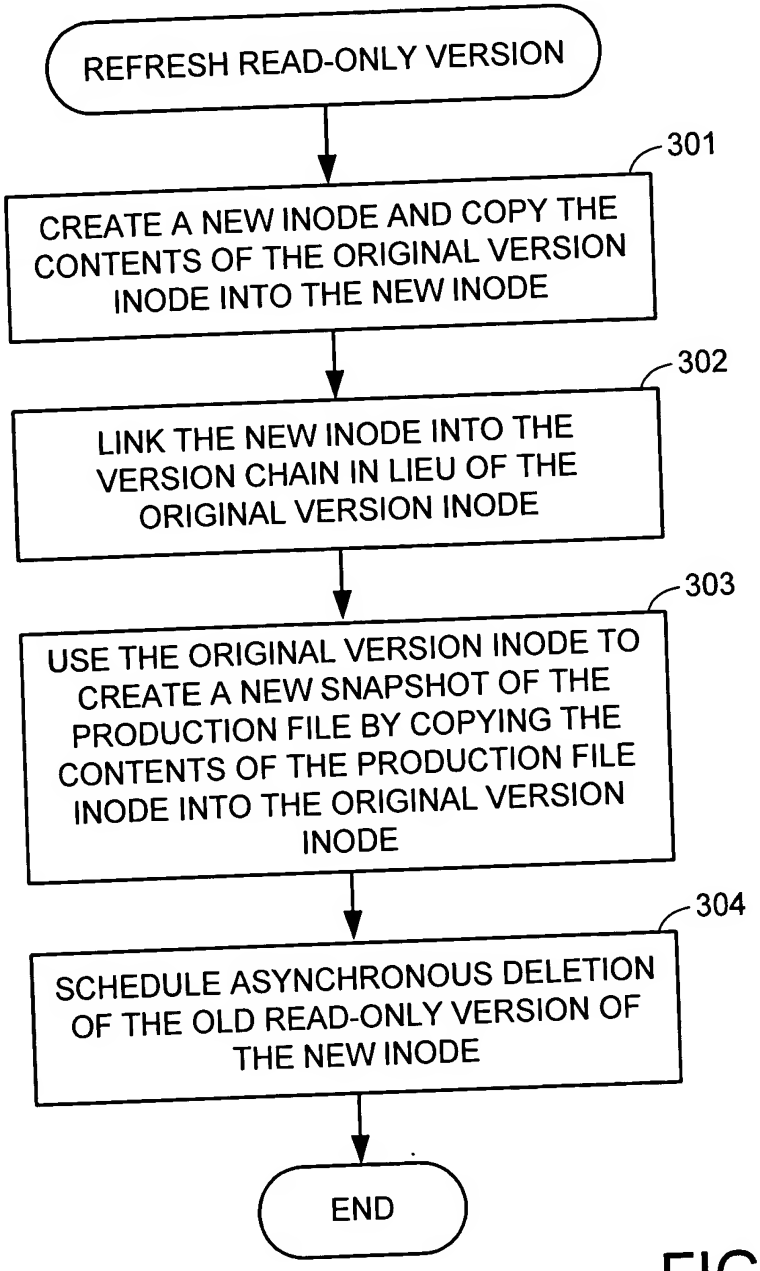

REFRESH READ-ONLY VERSION
301
CREATE A NEW INODE AND COPY THE CONTENTS OF THE ORIGINAL VERSION INODE INTO THE NEW INODE
302
LINK THE NEW INODE INTO THE VERSION CHAIN IN LIEU OF THE ORIGINAL VERSION INODE
303
USE THE ORIGINAL VERSION INODE TO CREATE A NEW SNAPSHOT OF THE PRODUCTION FILE BY COPYING THE CONTENTS OF THE PRODUCTION FILE INODE INTO THE ORIGINAL VERSION INODE
304
SCHEDULE ASYNCHRONOUS DELETION OF THE OLD READ-ONLY VERSION OF THE NEW INODE
END

FIG. 33

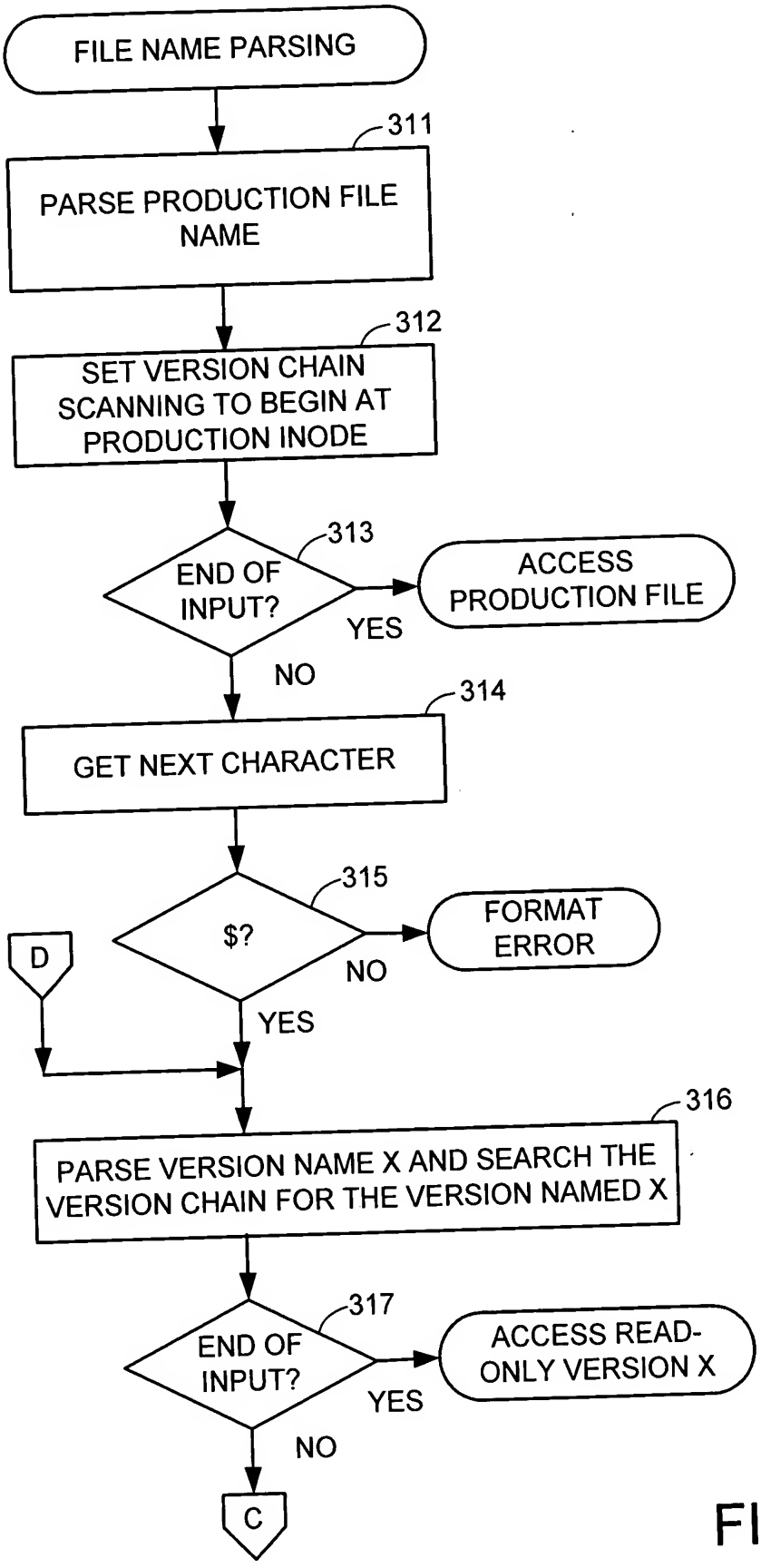

FILE NAME PARSING
311
PARSE PRODUCTION FILE NAME
312
SET VERSION CHAIN SCANNING TO BEGIN AT PRODUCTION INODE
313
END OF INPUT?
YES
ACCESS PRODUCTION FILE
NO
314
GET NEXT CHARACTER
315
$?
NO
FORMAT ERROR
D
YES
316
PARSE VERSION NAME X AND SEARCH THE VERSION CHAIN FOR THE VERSION NAMED X
317
END OF INPUT?
YES
ACCESS READ-ONLY VERSION X
NO
C

FIG. 34

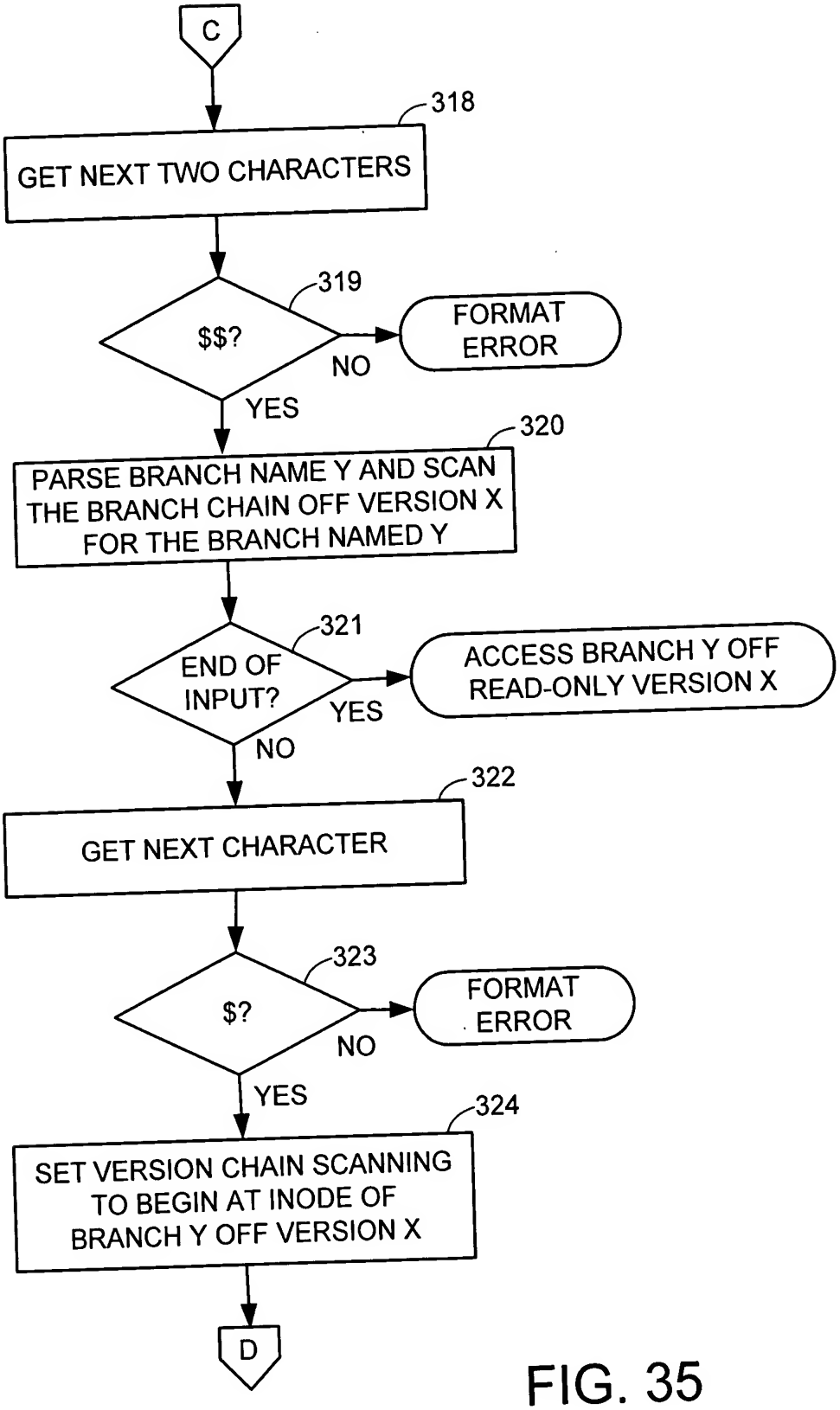

C
318
GET NEXT TWO CHARACTERS
319
$$?
NO
FORMAT ERROR
YES
320
PARSE BRANCH NAME Y AND SCAN THE BRANCH CHAIN OFF VERSION X FOR THE BRANCH NAMED Y
321
END OF INPUT?
YES
ACCESS BRANCH Y OFF READ-ONLY VERSION X
NO
322
GET NEXT CHARACTER
323
$?
NO
FORMAT ERROR
YES
324
SET VERSION CHAIN SCANNING TO BEGIN AT INODE OF BRANCH Y OFF VERSION X
D

FIG. 35